








ShuttleBus: Dense Packet Assembling With QUIC Stream Multiplexing for Massive IoT

Bo He , Member, IEEE, Jingyu Wang , Senior Member, IEEE, Qi Qi , Senior Member, IEEE, Qiang Ye , Senior Member, IEEE, Qihao Li , Member, IEEE, Jianxin Liao , and Xuemin Shen , Fellow, IEEE

Abstract—In this paper, we investigate dense short packet forwarding for clustering-based massive Internet-of-Things (mIoT). The objective is to support the data forwarding with minimal communication overhead while satisfying the differentiated latency constraints from the transport layer perspective. To this end, we propose a dense packet assembling scheme, named ShuttleBus, for forwarding devices in mIoT to achieve effective data merging. The assembling scheme is designed based on the stream multiplexing mechanism of the Quick UDP Internet Connection (QUIC) protocol. With ShuttleBus, the payload data sent from IoT devices are extracted as independent frames belonging to different data streams. The ShuttleBus can bundle data frames from multiple streams into a single packet while ensuring data integrity of these streams. Furthermore, we develop a resilient packing mechanism in packet assembling to merge data received from IoT devices within a cluster. In addition, a latency-oriented scheduling mechanism for backlogged QUIC data is established to guarantee satisfactory delivery of diverse transmission tasks. To accommodate the dynamic network environment, we tailor a learning-based algorithm to determine the optimal packet assembling time adaptively. We evaluate the performance of ShuttleBus under various network load conditions. Both analytical and experimental results demonstrate that the proposed scheme significantly reduces communication overhead and enhances data delivery performance under stringent latency constraints.

Index Terms—Packet assembling and scheduling, massive IoT, QUIC, stream multiplexing, deep reinforcement learning.

Manuscript received 25 May 2023; revised 8 November 2023; accepted 18 December 2023. Date of publication 25 December 2023; date of current version 2 July 2024. This work was supported in part by the National Natural Science Foundation of China under Grants U23B2001, 62171057, 62071067, and 62201148, in part by the National Postdoctoral Program for Innovative Talents under Grant BX20230052, in part by Guangdong Province Basic and Applied Basic Research Foundation under Grant 2022KQNCX, in part by the Key Area Research and Development Program of Guangdong Province under Grant 2020B0101110003, in part by China Postdoctoral Science Foundation under Grant 2023TQ0039, in part by the Ministry of Education and China Mobile Joint Fund under Grants MCM20200202, and MCM20180101, and in part by Beijing University of Posts and Telecommunications-China Mobile Research Institute Joint Innovation Center. Recommended for acceptance by V.P. Ranga Rao. (Corresponding authors: Jingyu Wang; Jianxin Liao.)

Bo He, Jingyu Wang, Qi Qi, and Jianxin Liao are with the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China (e-mail: hebo@bupt.edu.cn; wangjingyu@bupt.edu.cn; qiqi8266@bupt.edu.cn; liaojx@bupt.edu.cn).

Qiang Ye is with the Department of Electrical and Software Engineering, University of Calgary, Calgary, AB T2N 1N4, Canada (e-mail: qiang.ye@ucalgary.ca).

Qihao Li is with the College of Communication Engineering, Jilin University, Changchun 130012, China (e-mail: qihaol@jlu.edu.cn).

Xuemin Shen is with the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON N2L 3G1, Canada (e-mail: sshen@uwaterloo.ca).

Digital Object Identifier 10.1109/TMC.2023.3345898

I. INTRODUCTION

THE massive Internet-of-Things (mIoT), featuring densely deployed machine-type communication devices (MTCs), has empowered typical application scenarios of the next generation wireless networking, such as data dissemination in smart farming, remote control in industrial automation, and city-wide sensing and monitoring [1], [2]. The MTC density is anticipated to increase up to 10^7 per km^2 [3], and packet generation rates of MTCs usually vary depending on the supported IoT services [4]. The consistent transmission of short packets with varying packet generation rates is a distinctive characteristic of MTCs. These massive, but short packets dynamically sent from MTCs to a central base station (BS), present challenges in controlling packet transmission collisions [5].

Partitioning MTCs into clusters is a possible solution for collision mitigation in mIoT [6], [7]. As shown in Fig. 1, the MTC clusters with reduced contention regions maintain machine-type communication gateways (MTCGs) for data forwarding [8], [9]. Among existing clustering-based solutions, MTCGs employ the conventional TCP [10], [11] to forward the aggregated packets received from MTCs, which reduces the number of transmitted packets with larger sizes. Nevertheless, these approaches may not be suited for mIoT accommodating massive short packets, as a substantial portion of resources are dedicated to transmitting packet headers rather than user data. In this case, MTCGs continue to experience high communication overhead for data forwarding, and frequent network congestion arises due to increased packet transmission rates.

Aggregating received raw user data into a single packet at MTCGs has been effective in decreasing the communication overhead by eliminating recurring lower-layer headers. It also reduces latency by creating less interference on the shared spectrum [12]. However, it is crucial to ensure data integrity from different MTCs during the packet assembling process using a lossless approach. Here, data integrity is defined as the condition in which each received user data block from MTCs remains unaltered by others in the same forwarding packet. If data integrity is not preserved, it becomes difficult for a central BS in mIoT to recover the received data from MTCGs accurately. Also, the raw user data originating from various applications in mIoT are often heterogeneous and cannot be processed using traditional aggregation schemes commonly used in wireless sensor networks (WSNs) [13], [14], [15]. A potential solution is

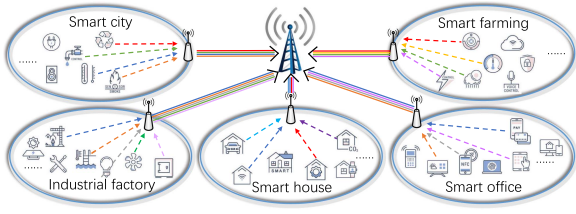


Fig. 1. Transmission architecture supporting massive short packets sent by MTCDs of various services in clustering-based mIoT.

to apply stream multiplexing, which enables the establishment of parallel data streams with data packing. In particular, the stream control transmission protocol (SCTP) [16], [17] is a transport protocol that supports a stream multiplexing mechanism and can be implemented at MTCGs to achieve data merging. However, adopting SCTP in mIoT scenarios with dynamic packet transmissions is challenging since the SCTP only allows for the multiplexing of a fixed number of streams [18]. Therefore, proposing an effective packet assembling and forwarding scheme for clustered-based mIoT scenarios is essential.

The stream multiplexing mechanism of the Quick UDP Internet Connection (QUIC) protocol [19] accommodates a varying number of data streams, with each stream having a distinct data byte sequence. Besides, the payload of a QUIC packet comprises frames that may carry user data or control signals from different streams [20]. Consequently, QUIC ensures the data integrity of streams while providing a tailored transmission strategy for each stream. Nevertheless, merging received short packets using QUIC at the MTCGs still faces the three following challenges:

- *Stream scheduling with differentiated and dynamic latency guarantee:* Various tasks of MTCDs have differentiated latency constraints [21], [22] (e.g., 1 ms latency in motion control and 50 ms latency in remote monitoring [23]). When heavy network congestion occurs, the transmission of tasks that require ultra-low latency can be throttled due to the simplified task scheduling and compulsory loss recovery in the existing QUIC protocol. Data deliveries with latency violations may cause severe safety hazards and economic losses [24], [25]. Therefore, for a complex and highly dynamic mIoT scenario, how to flexibly deliver tasks from different streams with various stringent latency requirements needs investigation.
- *Frequent packet assembling with diminishing gain:* The arrival of each packet at the MTCGs triggers the packet assembling in the existing QUIC protocol if the sender is idle. However, the efficiency of current packet assembling schemes [26] can still be improved by better utilizing the available space of assembled packets. Consequently, fewer packets are merged in the forwarding process, and the multiplexing gain of communication overhead is diminished.
- *Trade-off between merging efficiency and latency:* Although packet merging at the MTCGs significantly reduces the communication overhead, it inevitably brings additional latency for merging the data [27]. The forwarding performance of each MTCG is determined by the real-time network load conditions and its individual processing

capability. Consequently, it is challenging to obtain the lowest communication overhead under latency constraints in different mIoT environments.

In this paper, we design a self-adaptive packet assembling scheme, called *ShuttleBus*,¹ based on the QUIC protocol to merge the massive short received packets at MTCGs. Specifically, an MTCG multiplexes concurrent data streams originating from multiple MTCDs and extracts the received user data from these streams as frames in QUIC packets. Subsequently, The MTCG forwards the merged packets over a unique connection to a central BS. *To address the first challenge*, we customize a new type of QUIC frame to flexibly and rapidly notify the peer of the latest transmission requirements of a specific stream. We then establish latency-oriented stream-level transmission control to meet the requirements of each stream, including sending priorities, deadline-based scheduling, and optional retransmission. *To address the second challenge*, we devise a resilient mechanism for the QUIC packet assembling process. This mechanism allows the assembling packet to wait for the possible arrivals of new frames, provided that the packet still possesses adequate payload space. As a consequence, this mechanism further increases the number of multiplexing frames, thereby reducing the number of forwarding packets. *To address the third challenge*, we design a learning-based algorithm to limit the tolerance time for each QUIC assembling packet according to the real-time network conditions. In this way, the MTCGs can balance the tradeoff between merging efficiency and additional latency incurred during packet assembling, ultimately deriving optimal performance.

We implement ShuttleBus at the MTCGs and conduct extensive experiments under various network load conditions. At the premise of forwarding the same amount of user data, ShuttleBus reduces the packet rate by over 90%, decreases the total data rate by an average of 30% and up to 45%, and reduces the number of timeout data frames by an average of 70% and up to 80%. Our main contributions are four-folded:

- We design ShuttleBus, a self-adaptive packet assembling scheme at MTCGs based on the stream multiplexing of QUIC. It significantly reduces the communication overhead and thus increases the supported MTCD number under the central BS.
- We build a latency-oriented negotiation and scheduling mechanism to address the time-varying and differentiated latency constraints of streams. To satisfy these constraints, multiple transmission rules are provided for each stream.
- We develop a resilient packet assembling mechanism to further increase the multiplexing frame number in packets and alleviate congestion at MTCGs.
- We devise a learning-based algorithm to determine the tolerance time for the current assembling packet. This algorithm adapts to real-time network load and backlogged data conditions.

The rest of this paper is organized as follows. Section II presents a review of the relevant literature works. Section III

¹In our scheme, an assembled packet forwards the user data from multiple received packets like a shuttle bus transports multiple passengers simultaneously.

provides an overview of the mIoT scenario and the packet assembling process. The proposed packet assembling scheme, ShuttleBus, is presented in Section IV, and the performance analysis of the proposed scheme design is presented in Section V. Experimental results are discussed in Section VI. Section VII draws the concluding remarks for the paper.

II. RELATED WORK

Traffic packets in mIoT networks are characterized by their large quantity and short lengths. In order to manage the massive packets, many strategies have been investigated within the field of data aggregation. Based on the specific aggregation function employed, we present a comprehensive overview of existing schemes, delineating those that utilize either a lossy or lossless approach, respectively.

A. Data Aggregation in a Lossy Approach

Data aggregation is a widely recognized technique for enhancing energy efficiency by minimizing communication overhead in WSNs [28], [29]. This scheme aggregates multiple data items from information sources (i.e., sensors) to sink(s) to prolong the network's operational lifetime [30]. Data aggregation schemes in WSNs can be broadly categorized into two groups: temporal and spatial solutions [15]. Temporal solutions promote packet convergence over time and aggregate sensor data items using customized aggregation functions [13]. Solis et al. [31] employed a fixed rooting tree where elected nodes aggregate the packets from their children. The nodes are required to wait for a duration dependent on their depth in the tree to achieve the lowest communication overhead. To maximize aggregation benefits, Fan et al. [32] proposed a randomized waiting strategy that introduces artificial latency to packets, thereby enhancing temporal convergence. Spatial solutions, on the other hand, aim to design an optimal routing protocol based on the spatial relationships of data. In a clustering-based structure, a cluster head aggregates data from all sensors within this cluster and directly forwards a concise digest to the sink. Heinzelman et al. [33] assumed all nodes possess equal energy capacity and developed the LEACH protocol, which elects a cluster head based on real-time network conditions. Younis et al. [14] targeted to maximize network lifetime and proposed the HEED protocol to form efficient clusters, assuming the availability of multiple power levels at sensor nodes.

In previous studies, communication overhead was decreased by performing intermediate node processing on raw data using operations like MAX, MIN, SUM, and AVG. Consequently, only the abstracted data rather than raw data were forwarded to the sink.

Despite processing homogeneous raw readings is essential for reducing communication load, thereby enhancing the network's efficiency, transmitting all raw readings may result in significant energy consumption and communication overhead, potentially limiting the effective network throughput. However, processing homogeneous raw readings (such as temperature) can lead to a loss of precision compared to transmitting all raw readings [15]. Therefore, striking a balance between minimizing transmission

costs and preserving data accuracy is crucial for further improving network performance.

B. Data Aggregation in a Lossless Approach

Compared with the cluster heads in WSNs, the MTCGs in clustering-based mIoT have a significantly larger number of received packets and connecting MTCDs. Most existing studies on data aggregation in clustering-based mIoT focused on transmission schemes rather than aggregation functions to support massive readings. To characterize the interference and coverage performance for massive Machine-Type Communications (mMTC) with data aggregation, Guo et al. [34] built a tractable two-hop transmission model and obtained key performance metrics such as MTCD success probability. Wang et al. [7] analyzed the joint queue-length evolution of MTCGs and employed a transmission scheme that forwards stored data packets only when the length reaches a predetermined threshold. López et al. [35] designed principles based on non-orthogonal multiple access (NOMA) to enable data aggregation at MTCGs. In their proposed hybrid access scheme, several MTCDs can share the same orthogonal channel. Kim et al. [36] concentrated on reducing signaling overhead caused by packet aggregation and determined the optimal number of aggregators as a function of MTCD density. Akyurek et al. [12] considered the various applications in an mIoT cluster and reduced the number of packets by aggregating data sent from multiple applications on individual MTCDs instead of MTCGs. They traded off the aggregation gain and latency by designing a gain function for the optimal sending period.

The existing studies in this field predominantly rely on the assumption that the raw data from all MTCDs and applications can be aggregated into a single packet, and subsequently transmitted to sinks (e.g., the central BS). These studies employ lossless aggregation approaches, promising that all readings can be properly reconstructed from their aggregates at the sinks [29]. These approaches result in savings in the number of packets, however, the total number of bytes being transmitted remains unchanged. Notably, the present lossless approaches are insufficient in addressing the challenges posed by the current mIoT systems. These challenges stem from the fact that the amount of data transferred is continually increasing, leading to elevated energy consumption and network congestion. Therefore, it is important to explore alternative approaches that can efficiently reduce both the amount of transferred bytes and number of packets, without sacrificing the accuracy of the raw transmitted data.

C. Summary

Different from the aforementioned studies, our work accomplishes data aggregation at MTCGs in a lossless approach, which concurrently reduces both the number of packets and bytes sent. Our primary focus lies in the practical implementation of the complex packet merging tasks within the transport layer, while accommodating multiple applications through the stream multiplexing mechanism facilitated by the emerging QUIC protocol.

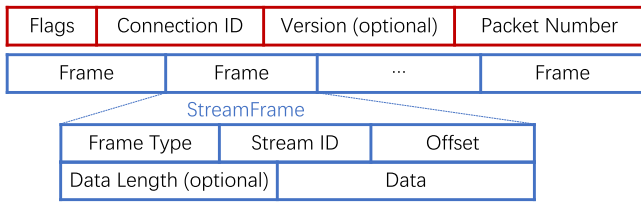


Fig. 2. Typical QUIC packet structure and a *StreamFrame* structure.

Besides, ShuttleBus quantitatively evaluates the gains of communication overhead reduction resulting from data aggregation in mIoT scenarios.

III. NETWORKING SCENARIO AND MOTIVATION

A. Clustering-Based mIoT Scenario

Consider a clustering-based mIoT network comprising multiple MTCGs that connect hundreds of MTCs, as Fig. 1 shows. The clusters can be built and managed based on various factors, such as latency [37], location [38], and traffic characteristics [39]. In this paper, we adopt the location-based clustering scheme to construct the mIoT network and our work is compatible with all these clustering schemes. This clustering-based network builds a two-hop transmission architecture where MTCs access is divided into an intra-clustering transmission phase and an MTCG forwarding phase. During the intra-clustering transmission phase, MTCs use lower power to send short packets to their associated MTCGs rather than the central BS. In the MTCG forwarding phase, MTCGs disassemble the received packets, reassemble new packets, and forward these new packets to the BS. In the mIoT network, densely deployed MTCs, such as sensors, actors, and controllers, have differentiated latency constraints to execute various tasks [40]. Moreover, the constraints of some emerging services, which necessitate ultra-low latency, such as haptic communication, may decrease to a sub-millisecond level [41].

Except for the differentiated transmission requirements, there are some prominent communication characteristics in mIoT: (i) short packets: packet payload lengths range from a few bytes to several hundred bytes [36]; and (ii) uplink-dominated transmission: a substantial portion of the data traffic is attributed to sensor readings or MTC status reports [42]. Given these characteristics, MTCGs can potentially reduce communication overhead by efficiently merging the short received uplink packets.

B. Packet Assembling in QUIC

As illustrated in Fig. 2, the QUIC packet structure comprises a public header (colored in red) and a payload (colored in blue) composed of a varying number of frames. Each frame serves as an independent entry that carries either user data or control signals. The frame type responsible for carrying user data is referred to as *StreamFrame*, and its *Stream ID* field indicates the stream to which it belongs. Consequently, a QUIC packet can accommodate frames from multiple streams without

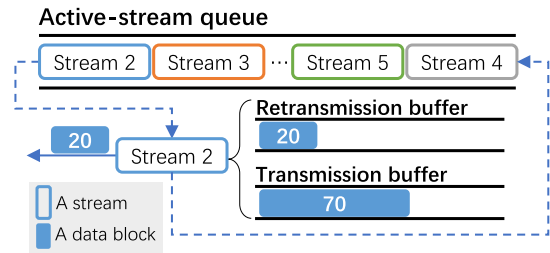


Fig. 3. Default active-stream queue and the frame packing rule of QUIC.

conflict [43]. Besides, each stream possesses an exclusive data sequence, and the *Offset* field of each *StreamFrame* signifies the position of its carried user data within the corresponding sequence. Upon receiving a packet containing multiple *StreamFrames*, the receiver disassembles the packet and segments the entire payload into *StreamFrames* by interpreting the values in their *Data Length* fields. Therefore, the stream multiplexing of QUIC ensures data integrity for each stream during transmission.

Based on the packet structure, Fig. 3 demonstrates the packet assembling process of QUIC. Specifically, the sender maintains an active-stream queue for all streams with data awaiting transmission. When initiating packet assembling, the sender requests data from these active streams sequentially. Each requested stream is popped from the queue and a data block is taken either from the stream's retransmission buffer or from its transmission buffer. This data block will be encapsulated as a *StreamFrame* with a specific offset. If there is still data to be transmitted, the stream will be appended to the end of the active-stream queue. For example, as shown in Fig. 3, Stream 2 takes a data block with an offset being 20 from its retransmission buffer and it will append the data with the offset being 70 to the end of the active-stream queue. The derived *StreamFrames* are then packed into the current packet and prepared for transmission. Two possible scenarios may lead to the end of packet assembling: (i) the remaining payload space of the current packet is insufficient to accommodate additional frames, or (ii) all active streams have been processed.

The packet assembling process of QUIC exhibits two distinct characteristics: (i) if the current active-stream queue is empty, packet assembling is immediately triggered by the data arrival in any stream, and (ii) the packing order of *StreamFrames* strictly adheres the sequence of active streams without any additional priority rules.

C. Our Approach

Integrating packet assembling with stream multiplexing at the MTCGs offers several advantages for mIoT scenarios: (i) the payload data of most packets sent by MTCs comprise only tens of bytes, whereas the maximum length of a QUIC packet is approximately 1,500 bytes. This disparity enables MTCGs to consolidate the data from multiple received packets into a single outgoing packet; (ii) MTCs generate a vast number of parallel data streams, facilitating the assembling of packets with data from distinct MTCs utilizing stream multiplexing;

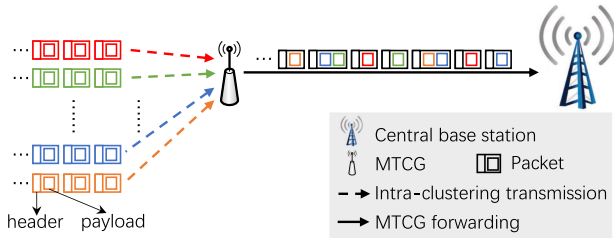


Fig. 4. Packet merging process based on the stream multiplexing mechanism at an MTCG.

and (iii) the volume of waiting data in a single active stream is insufficient to fill an empty packet, given the low transmission data rates of most MTCGs. Consequently, scheduling queued active streams can efficiently manage packet assembling in the considered mIoT scenario.

Leveraging these advantages, we aim to design a packet assembling approach that efficiently merges the received packets while adhering to their latency constraints. Initially, we recognize the heterogeneous latency constraints of streams and establish tailored packet assembling rules for each. In this manner, the unique constraints of streams are satisfied to the greatest extent possible during packet assembling. Subsequently, we strive to minimize communication overhead by increasing the average multiplexing frame count in assembled packets. Specifically, each partially filled QUIC packet is permitted to wait for a duration and accommodate newly arriving *StreamFrames* during this interval. Lastly, to prevent the excessive prioritization of merging efficiency at the expense of latency, we design an algorithm that constrains the tolerance assembling time of QUIC packets. This algorithm endeavors to achieve the lowest communication overhead under latency constraints of raw data, depending on real-time network load conditions and historical forwarding capabilities.

IV. DENSE PACKET ASSEMBLING SCHEME

The proposed dense packet assembling scheme, ShuttleBus, comprises four components: (i) a data merging approach based on QUIC stream multiplexing is designed for forwarding tasks at MTCGs; (ii) a latency-oriented negotiation and scheduling mechanism is developed for all active streams, allowing for flexible satisfaction of their latency constraints; (iii) a resilient packing mechanism is implemented for extracted frames, which serves to further reduce communication overhead; and (iv) an adaptive learning-based algorithm is formulated to adjust the tolerance assembling time of the current packet. These components are elaborated upon as follows.

A. Stream Multiplexing-Based Data Merging

As shown in Fig. 4, each MTCG establishes a connection with each neighboring MTCG and the central BS, utilizing the two-hop transmission architecture. Through these connections, MTCGs receive packets from MTCGs and forward the acquired user data to the central BS. We design the stream multiplexing-based data merging (SMDM) approach using the QUIC protocol.

Frame Type (8)	Stream ID (8)
Priority (8)	Retransmission rule (8)
Deadline (8)	

Fig. 5. Structure format of *StreamPropertyFrame* (in the unit of byte).

For an individual MTCG, the process involves the following steps:

- Establishing a dedicated stream for the received data of each connected MTCG in the connection with the BS.
- Disassembling the received packets to retrieve user data and storing this data in the buffers of their respective streams.
- Extracting available data blocks from active streams as *StreamFrames* and packing them into the current packet.
- Sending the assembled packet, consisting of a public header and all *StreamFrames*, to the central BS.

The SMDM approach enables an MTCG to multiplex concurrent data streams originating from distinct MTCGs, ensuring the data integrity of these streams during transmission. Compared with the conventional single-frame (data block) per packet forwarding approach, SMDM diminishes the number of forwarding packets by permitting the consolidation of data from multiple packets into a single one. As a result, the MTCG significantly reduces the number of bytes sent for packet headers. Besides, when the network topology changes because an MTCG joins or exits the cluster, the SMDM approach needs to add or delete a QUIC data stream in the connection between the MTCG and the central BS. In this way, this approach can cope with the dynamic network topology of the clustering structure well.

B. Latency-Oriented Negotiation and Scheduling

In the current QUIC packet assembling with stream multiplexing, several rules present challenges to latency guarantees: (i) the sender and receiver lack a flexible and rapid negotiation approach for latency constraints; (ii) the packing order of *StreamFrames* strictly adheres to the queuing sequence of active streams, preventing a stream from extracting its awaiting data until all preceding streams have been processed; and (iii) mandatory retransmission rules may increase latency and impede subsequent data transmission, for example, a lost *StreamFrame* should not be retransmitted if it has exceeded the latency constraints of its stream.

To satisfy the latest latency constraint of each stream, we build the latency-oriented negotiation and scheduling (LONS) mechanism that consists of two modules as follows:

1) *Negotiation Module*: To ascertain the transmission rules of each stream, the sender and the receiver require a negotiation method to exchange information. Leveraging the adaptable frame-based QUIC packet structure depicted in Fig. 2, we design and register a frame type named *StreamPropertyFrame* to convey the transmission rules for a specific stream. As shown in Fig. 5, a *StreamPropertyFrame* contains five fields:

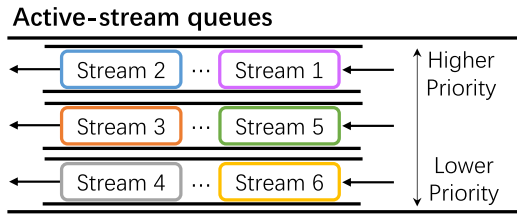


Fig. 6. Multi-priority active-stream queues. Stream 5 has a data block that needs to be retransmitted.

- **Frame Type:** The identifier of this frame type. We register 0×20 as the *StreamPropertyFrame* type's identifier.
- **Stream ID:** The identifier of the stream subject to latency negotiation. The value is unique within a connection.
- **Priority:** The sending priority of the stream, with a default value of 0. A higher value represents increased priority.
- **Retransmission Rule:** The retransmission rule governing the stream. The default value is 0, signifying that a lost frame associated with this stream must be retransmitted. When set to 1, the decision to retransmit a lost frame depends on the elapsed time and deadline constraint. When set to 2, the lost frame will not be retransmitted.
- **Deadline:** The latency constraints applicable to the stream. The default value is 0, indicating that the stream does not have stringent latency constraints. When assigned a positive value, the latency constraint is expressed in milliseconds.

By sending *StreamPropertyFrames*, the sender can flexibly and rapidly notify the receiver of these transmission rules when a stream is established or when its rules are updated dynamically based on its corresponding applications.

2) **Scheduling Module:** According to the latest information received from the *StreamPropertyFrames*, we establish two scheduling rules for active streams to ensure their latency constraints, as detailed below:

- **Sending priority rule:** We assign a priority level (indicated in the *Priority* field) to each stream within the same connection and build multiple active-stream queues. As shown in Fig. 6, each priority possesses a dedicated active-stream queue. If a stream has data to send, it is pushed into the active-stream queue corresponding to its assigned priority. During packet assembling, the sender requests data from streams with higher priority levels before addressing those with lower priority levels. In this way, the streams with higher priority levels send their awaiting data as soon as possible, which helps satisfy their latency constraints when congestion arises at the MTCG.
- **Retransmission rule:** We tailor a specific retransmission rule (indicated in the *Retransmission Rule* field) for each stream. For certain streams with ultra-low latency constraints that cannot accommodate retransmission latency, the lost *StreamFrames* may be designated as non-retransmittable by the sender. Conversely, for streams with more relaxed latency constraints, their lost *StreamFrames* may be retransmitted as long as doing so does not breach

the constraints. It should be noted that any optional retransmission necessitates adjustments to the relevant flow control rules for both the sender and receiver. For the sake of brevity, we do not discuss these modifications in detail within this paper.

C. Resilient Frame Packing

To assemble a packet, the sender must scan the active-stream queues, extract the available *StreamFrames*, and pack them into the current packet. If there are no active streams in the queues, the sender periodically scans the queues until it obtains at least one *StreamFrame*. Once *StreamFrames* start to be packed into the current packet, the sender stops scanning until the packet has loaded all extracted *StreamFrames*. Under the existing assembling mechanism, when peak traffic is absent, most packets only contain one or two *StreamFrames*. The reason is that the scanning period for the active-stream queues is too brief to coincide with the arrivals of multiple new packets sent from MTCs. Accordingly, the proposed SMDM approach does not demonstrate apparent advantages over the conventional forwarding approach without data merging in terms of communication overhead.

To further enhance the benefits of data merging, we propose the resilient frame packing (RFP) mechanism to increase the average number of *StreamFrames* in each assembled forwarding packet. In RFP, after loading all extracted *StreamFrames*, the current packet waits for a resilient period if it has sufficient free payload space to load additional *StreamFrames*, instead of being sent immediately. During the waiting period, the sender continues to scan the active-stream queues and pack the extracted *StreamFrames* from newly arrived packets into the current packet. We set a tolerance time (denoted by T) to limit the total frame packing time of a single packet. The algorithm for calculating the tolerance time will be elaborated in Section IV-D. In this way, the packet will be sent after the tolerance time or when filled with *StreamFrames*. As a result, there are three possible cases in the packet assembling process:

- **Case 1:** Initial *StreamFrames* do not fill the packet, and it waits for time T but without loading enough *StreamFrames*.
- **Case 2:** Initial *StreamFrames* do not fill the packet, and it waits for time t ($t \leq T$) to load enough *StreamFrames*.
- **Case 3:** Initial *StreamFrames* fill the packet, and it is sent immediately after loading enough *StreamFrames*.

D. ShuttleBus Algorithm

Although the RFP mechanism significantly reduces communication overhead and alleviates congestion, it inevitably brings additional latency. Consequently, a balance has to be struck between frame multiplexing efficiency and data latency during packet assembling. In the packet assembling process with RFP, the average latency of *StreamFrames* is positively correlated with the tolerance assembling time T (will be proved in Section V-B). Based on this observation, we tailor the *ShuttleBus* algorithm to adaptively determine an appropriate T for each

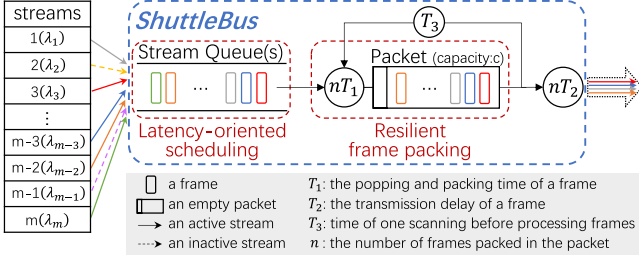


Fig. 7. Workflow of ShuttleBus at an MTCG.

packet. This algorithm calculates the maximum T under latency constraints, achieving the lowest communication overhead.

1) *System Modeling*: As shown in Fig. 7, the QUIC packet assembling process at the MTCGs comprises data arrivals from MTCs, packet assembling, packet sending, and periodic scanning of the active-stream queues for newly arrived data.

Similar to existing works [12], [36], we model packet arrivals using a Poisson process in an mIoT environment. This selection is justified for two primary reasons: first, employing Poisson process-based modeling enables a more tractable performance analysis; second, the arrival characteristics observed in actual massive-access tests demonstrate a strong correlation with these models generated by the Poisson process. Concretely, for an MTCG connecting with N MTCs, packet arrivals at the MTCG from MTC i are modeled as a Poisson process with arrival rate λ_i . Then, the packet arrivals from all MTCs form a Poisson process with the total arrival rate $\Lambda = \sum_{i=1}^N \lambda_i$. We define the packing time of a *StreamFrame* as a constant T_1 , which includes the time required to extract it from a stream and configure its frame header. The transmission time of a *StreamFrame* is a constant T_2 when the bandwidth of the link remains stable. Upon completing the assembling of the current packet j , the sender scans the active-stream queues for newly arrived *StreamFrames* to assemble the subsequent packet $j+1$. The scanning interval for the queues is also a constant T_3 . The constant c represents the maximum number of *StreamFrames* that can be packed into an empty packet, which is determined by the average length of the arrived *StreamFrames*. In the default packet assembling process without RFP, for a packet containing n ($n \leq c$) *StreamFrames*, the maximum experienced latency D of its *StreamFrames* is:

$$D = t_b + nT_1 + nT_2 \quad (1)$$

where t_b ($t_b \leq T_3$) is the backlogged time of the first *StreamFrame* in the active-stream queues.

2) *CalMinRemPackTime() Function*: To derive the maximum value of T for each packet, we must calculate the remaining tolerance time of each packed *StreamFrame* according to its latency constraint and the current time.

More specifically, for the assembling packet j , we denote the set of *StreamFrames* to be packed into it as SF_j . Upon initiating the assembling of packet j , we calculate its tolerance assembling time T_j as:

$$T_j = \min \left(\delta_{SF_j, \min}, \frac{c}{\Lambda} - T_3 \right) \quad (2)$$

Algorithm 1: CalMinRemPackTime() Function.

Input: $\delta_{SF, \min}$, *StreamFrame* s , the current time t_0 , c , T_1 , T_2 , α .

Output: The minimum available packing time $\delta_{SF, \min}$.

- 1 $\sigma_s \leftarrow \text{GetLatencyConstraint}(s)$;
 - 2 $t_{s,0} \leftarrow \text{GetArrivedTime}(s)$;
 - 3 $t_{s,q} \leftarrow t_0 - t_{s,0}$;
 - 4 $\delta_s \leftarrow \alpha [\sigma_s - t_{s,q} - c * (T_1 + T_2)]$;
 - 5 **if** $\delta_s < \delta_{SF, \min}$ **then**
 - 6 $\delta_{SF, \min} \leftarrow \delta_s$;
 - 7 **return** $\delta_{SF, \min}$.
-

where $\delta_{SF_j, \min}$ is defined as the smallest remaining assembling time among all *StreamFrames* of SF_j . Given that the total assembling time includes both queues scanning and subsequent waiting, $\frac{c}{\Lambda} - T_3$ serves as a constraint that ensures T can effectively regulate the packet assembling process (will be explained in Section V-A). Each time a new *StreamFrame* is added to SF_j , we update the value of $\delta_{SF_j, \min}$ accordingly. For each *StreamFrame* s ($s \in SF_j$), we designate its remaining packing time as:

$$\delta_s = \alpha [\sigma_s - t_{s,b} - c * (T_1 + T_2)] \quad (3)$$

where σ_s is the latency constraint of s and $t_{s,b}$ is the backlogged time of s in the active-stream queues. Algorithm 1 exhibits the workflow of the *CalMinRemPackTime()* function that calculates the latest $\delta_{SF_j, \min}$. To avoid timeouts caused by potential network fluctuation, we allocate time for packing and sending c *StreamFrames* as shown in (3). Additionally, we employ the weighting factor α ($\alpha \in (0, 1]$) to balance the tradeoff between merging efficiency and latency of *StreamFrames*.

3) *Network Load-Aware Weighting Algorithm*: Calculating the optimal value of α is challenging in dynamic mIoT networks. Intuitively, for packet assembling within the same MTCG, a larger α value leads to reduced communication overhead but increased additional latency. However, our testing indicates that the performance of α is determined not only by the capability of the physical MTCGs but also by the real-time network load conditions.

Under heavy network load, a larger α achieves lower communication overhead and similar latency performance compared with a smaller α . This outcome occurs because a significant proportion of packets exit the waiting process early in *Case 2*, which assists in coping with potential network fluctuations. Conversely, under light network load, although a small α incurs higher communication overhead, it exhibits better latency performance than a large α . This finding is attributed to the fact that most packets cannot load enough *StreamFrames* in *Case 1* and consequently wait for the complete tolerance time. If unpredictable network fluctuations occur, the *StreamFrames* contained in these packets are more likely to violate latency constraints.

Therefore, the packet assembling scheme has to obtain the mapping from the network load degrees to the optimal α value. But for MTCGs with varying capabilities, the mappings also

Algorithm 2: NLaw Algorithm.

Input: agent A , total epoch number M .
Output: The optimal decision-making policy P .

- 1 Initialize: initial decision-making policy P , experience pool EP , $m \leftarrow 0$;
- 2 **while** $m < M$ **do**
 - 3 $p_m, d_m \leftarrow \text{GetEnvStates}(m)$; */
 - 4 $s_m \leftarrow \text{AssembleStateVector}(p_m, d_m)$;
 - 5 $a_m \leftarrow \text{DetermineAction}(A, s_m)$;
 - 6 $r_m \leftarrow \text{CalculateReward}(a_m)$; */
 - 7 $p_{m+1}, d_{m+1} \leftarrow \text{GetEnvStates}(m+1)$;
 - 8 $s_{m+1} \leftarrow \text{AssembleStateVector}(p_{m+1}, d_{m+1})$;
 - 9 $T_m \leftarrow \text{AssembleTransitionVector}(s_m, a_m, r_m, s_{m+1})$;
 - 10 $EP \leftarrow \text{AddExperience}(EP, T_m)$;
 - 11 $P \leftarrow \text{OptimizePolicyByTD3}(P, EP)$;
 - 12 $m \leftarrow m+1$;
- 13 **return** P .

differ. In this case, for each individual MTCG, we employ deep reinforcement learning (DRL) [44] to learn the mapping from the network load degrees to the optimal α value. Owing to the model-free learning capability, most DRL-based methods perform well in a Markov decision process (MDP) framework [45], [46], which involves an agent, state space, and action space. At each epoch, the agent observes the environment's states and determines a corresponding action. After executing this action in the environment, the agent receives a reward for the action to optimize its decision-making policy. Here, the policy represents a mapping from the state space to the action space, iteratively optimized by specific DRL optimization algorithms. We customize the MDP framework for forwarding tasks at MTCGs as follows:

- *Agent*: The agent is an entity deployed to an MTCG and makes decisions about the optimal α value for this MTCG.
- *State Space*: The state space needs to represent the real-time network load degrees of the MTCG. Hence, we select two metrics as the states: the normalized real-time packet rate and data rate received from MTCs.
- *Action Space*: Each action corresponds to an α value, making the action space a continuous range from 0 to 1.
- *Reward*: The reward of an action at a given epoch has to consider latency constraints and communication overhead. Thus, we define the reward function at epoch m as:

$$r_m = -(p_m + d_m) \quad (4)$$

where p_m and d_m represent the normalized proportion of timeout *StreamFrames* and the sent data amount from the MTCG at epoch m , respectively.

Using this framework, we can derive the optimal mapping from network load degrees to the weighting factor α value by optimizing the decision-making policy. Due to the continuous state and action spaces, we adopt the twin delayed deep deterministic (TD3) [47] algorithm that excels in continuous decision-making

Algorithm 3: ShuttleBus Algorithm.

Input: $\Lambda, c, T_1, T_2, T_3, \alpha$, active-stream queues Q .
Output: The packet j ready to be sent.

- 1 Initialize: packet j being assembled, $\delta_{SF_j, \min} \leftarrow \frac{c}{\Lambda} - T_3, SF_j \leftarrow \phi, n \leftarrow 0$.
- 2 **while** $|SF_j| == 0$ **do**
 - 3 $SF_j \leftarrow \text{GetStreamFrames}(Q, j)$.
 - 4 **if** $|SF_j| \geq c$ **then**
 - 5 $n \leftarrow n + 1$; */
 - 6 $j \leftarrow \text{PackStreamFrame}(j, SF_j[n])$;
 - 7 $n \leftarrow n + 1$
 - 8 **Return** packet j .
 - 9 **else if** $|SF_j| > 0$ **then**
 - 10 $t_0 \leftarrow \text{GetCurrentTime}()$;
 - 11 **for** *StreamFrame* s in SF_j **do**
 - 12 $j \leftarrow \text{PackStreamFrame}(j, s)$;
 - 13 $\delta_{SF_j, \min} \leftarrow \text{CalMinRemPackTime}(\delta_{SF_j, \min}, s, t_0, c, T_1, T_2, \alpha)$
 - 14 $t_j \leftarrow \text{GetCurrentTime}()$;
 - 15 **while** $(t_j - t_0 < \delta_{SF_j, \min})$ and $(|SF_j| < c)$ **do**
 - 16 $s', SF_j \leftarrow \text{GetNewArrivedSF}()$; */
 - 17 $\delta_{SF_j, \min} \leftarrow \text{CalMinRemPackTime}(\delta_{SF_j, \min}, s', t_j, c, T_1, T_2, \alpha)$;
 - 18 $j \leftarrow \text{PackStreamFrame}(j, s')$;
 - 19 $t_j \leftarrow \text{GetCurrentTime}()$;
 - 20 **Return** packet j .

tasks. To obtain a strong learning capability, all models including actors, critics, and target neural networks are built by three fully-connected neural layers. These models of a single TD3 agent are deployed on each MTCG and work for the respective MTCG. We outline the drafted DRL-based network load-aware weighting (NLaw) algorithm in Algorithm 2. As shown, at each epoch, the agent assembles a transition vector and stores it in the experience pool. The agent then samples some transition vectors from the experience pool and optimizes the policy using the TD3 algorithm. In this way, the agent can ultimately determine the optimal α value according to the real-time network load degrees for the MTCG. Note that for a specific MTCG, its computing capability and the total bandwidth between it and the central BS hardly change. Thus, the DRL models of an MTCG do not need to be frequently retrained using new data samples, and the model inference is usually time and computation-efficient.

4) *Complete Shuttlebus Algorithm*: In summary, the complete workflow of the *ShuttleBus* algorithm is outlined in Algorithm 3. For the current assembling packet j , the number of available *StreamFrames* in all active-stream queues is initially checked. Following this, as many *StreamFrames* as possible are packed into packet j . If there is still free space for additional

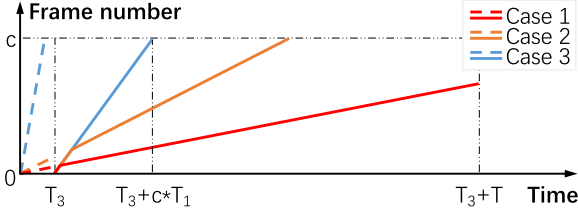


Fig. 8. Three possible cases of packet assembling.

StreamFrames, packet j starts to wait, and its current tolerance assembling time T_j is calculated based on the *CalMinRemPackTime()* function and the *NLAW* algorithm. During the waiting, the value of T_j is updated each time a new *StreamFrame* arrives. Either the time is up or packet j is full, the waiting is terminated, and the packet is sent immediately.

V. PERFORMANCE ANALYSIS

A. Impact of the RFP on Congestion

To ensure the long-term effective operation of MTCGs, we assume that the long-term average speed of data forwarding by MTCGs exceeds the average speed of data arrival. According to the queueing model constructed in Section IV-D, there must be a stable probability distribution π_n for n *StreamFrames* in an assembled packet. For each *StreamFrame*, violation of the latency constraint occurs solely due to congestion. In the packet assembling process without RFP, we define congestion as the arrival of at least c packets at the MTCG within a short time. Specifically, this short time encompasses the frame packing time of the last packet and the subsequent scanning interval for the current packet. For simplicity, we ignore the packet disassembling process and directly use the extracted *StreamFrames* to replace the received packets in the subsequent analysis. Given that arrivals of *StreamFrames* from all MTCGs adhere to a Poisson process, the probability of congestion P_C is expressed as:

$$P_C = \sum_{k=c}^{\infty} \sum_{n_0=0}^{c-1} \frac{\Lambda^k (T_3 + n_0 T_1 \pi_{n_0})^k}{k!} e^{-\Lambda(T_3 + n_0 T_1 \pi_{n_0})} \quad (5)$$

where n_0 is the number of *StreamFrames* in the last packet.

With RFP, we can model the three cases of packet assembling discussed in Section IV-C, as illustrated in Fig. 8. In this figure, the dotted lines represent the number of *StreamFrames* arrived, while the solid lines represent the number of *StreamFrames* packed. The cases exhibit distinct characteristics of the *StreamFrames*' arrival process as follows:

- *Case 1*: Fewer than c *StreamFrames* arrive during $(T + T_3)$.
- *Case 2*: Fewer than c *StreamFrames* arrive during T_3 , and at least c *StreamFrames* arrive during $(T + T_3)$.
- *Case 3*: At least c *StreamFrames* arrive during T_3 .

To ensure that the T setting effectively regulates RFP, $(T + T_3)$ should not exceed c/Λ ; otherwise, most packets will load c *StreamFrames* during T . Consequently, the congestion probability with RFP is equal to the probability of *Case 3*

occurrence:

$$P_3 = \sum_{k=c}^{\infty} \frac{\Lambda^k T_3^k}{k!} e^{-\Lambda T_3} \quad (6)$$

The conclusion that P_3 is smaller than P_C is reached regardless of the value of distribution π_n , which proves that RFP reduces the probability of congestion at the MTCG.

B. Average Latency of StreamFrames at the MTCGs

The packet assembling process with RFP constitutes a batch-service queueing process [48], [49]. The frame packing duration depends on the number of *StreamFrames* [50], but the total time of packet assembling varies (i.e., awaiting additional incoming *StreamFrames*). This attribute complicates the modeling process using conventional queueing models. Thus, we analyze the average latency of *StreamFrames* according to their arrival patterns.

Building upon the analysis presented in Section V-A, we initially derive the probabilities of *Case 1* and *Case 2* occurrences:

$$P_1 = \sum_{k=1}^{c-1} \frac{\Lambda^k (T + T_3)^k}{k!} e^{-\Lambda(T+T_3)} \quad (7)$$

$$P_2 = \sum_{k=c}^{\infty} \left[\frac{\Lambda^k (T + T_3)^k}{k!} e^{-\Lambda(T+T_3)} - \frac{\Lambda^k T^k}{k!} e^{-\Lambda T} \right]. \quad (8)$$

As depicted in (5), P_3 is a constant independent of T , which implies that the sum of P_1 and P_2 is also a constant equal to $(1 - P_3)$. Next, we calculate the derivative of P_1 :

$$P_1' = -\Lambda e^{-\Lambda(T+T_3)} \frac{(\Lambda T + \Lambda T_3)^{c-1}}{(c-1)!}. \quad (9)$$

Accordingly, P_1' remains consistently negative. Therefore, with the increase of T , P_1 decreases and P_2 increases, which proves that T directly governs the ratio of P_1 to P_2 . In the following, we solely analyze the relationship between T and the weighted average latency $D(T)$ of the first *StreamFrame* packed into the current packet in *Case 1* and *Case 2*. $D(T)$ represents the sum of the processing latency (inclusive waiting and packing time) and transmission latency.

In *Case 1*, the weighted average processing latency $D_{1,p}(T)$ and transmission latency $D_{1,t}(T)$ are given by:

$$D_{1,p}(T) = P_1 \cdot (T + T_3) \quad (10)$$

$$D_{1,t}(T) = P_1 \cdot n T_2 = P_1 \cdot \Lambda (T + T_3) T_2. \quad (11)$$

In *Case 2*, the weighted average processing latency $D_{2,p}(T)$ and transmission latency $D_{2,t}(T)$ are obtained based on the probability density function of a Poisson process:

$$\begin{aligned} D_{2,p}(T) &= \int_{T_3}^{T+T_3} \Lambda e^{-\Lambda t} \frac{(\Lambda t)^{c-1}}{(c-1)!} dt \\ &= \Lambda^{c-1} c \left[\left(\sum_{k=1}^c \frac{T_3^k}{k!} + \frac{1}{\Lambda} \right) e^{-\Lambda T_3} \right] \end{aligned}$$

$$- \left(\sum_{k=1}^c \frac{T^k}{k!} + \frac{1}{\Lambda} \right) e^{-\Lambda T} \quad (12)$$

$$D_{2,t}(T) = P_2 \cdot cT_2. \quad (13)$$

Thus, the weighted average latency in *Case 1* and *Case 2* can be expressed as:

$$\begin{aligned} D(T) &= D_{1,p}(T) + D_{1,T}(T) + D_{2,p}(T) + D_{2,T}(T) \\ &= (1 + \Lambda T_2)(T + T_3) \sum_{k=1}^{c-1} \frac{\Lambda^k (T + T_3)^k}{k!} e^{-\Lambda(T+T_3)} \\ &\quad + \Lambda^{c-1} c \left[\left(\sum_{k=1}^c \frac{T_3^k}{k!} + \frac{1}{\Lambda} \right) e^{-\Lambda T_3} \right. \\ &\quad \left. - \left(\sum_{k=1}^c \frac{T^k}{k!} + \frac{1}{\Lambda} \right) e^{-\Lambda T} \right] + P_2 \cdot cT_2. \end{aligned} \quad (14)$$

We denote the three terms in (13) as follows:

$$\begin{aligned} f_1(T) &= (1 + \Lambda T_2)(T + T_3) \sum_{k=1}^{c-1} \frac{\Lambda^k (T + T_3)^k}{k!} e^{-\Lambda(T+T_3)} \\ f_2(T) &= \Lambda^{c-1} c \left[\left(\sum_{k=1}^c \frac{T_3^k}{k!} + \frac{1}{\Lambda} \right) e^{-\Lambda T_3} \right. \\ &\quad \left. - \left(\sum_{k=1}^c \frac{T^k}{k!} + \frac{1}{\Lambda} \right) e^{-\Lambda T} \right] \\ f_3(T) &= P_2 \cdot cT_2. \end{aligned} \quad (15)$$

Then, the derivative of the weighted average latency $D(T)$ can be obtained by adding the derivatives of the three terms. To this end, we calculate these derivatives respectively:

$$\begin{aligned} f_1'(T) &= (1 + \Lambda T_2) \left[\sum_{k=1}^{c-1} \frac{\Lambda^k (T + T_3)^{k+1}}{k!} e^{-\Lambda(T+T_3)} \right]' \\ &= (1 + \Lambda T_2) e^{-\Lambda(T+T_3)} \end{aligned} \quad (16)$$

$$\begin{aligned} f_2'(T) &= -\Lambda^{c-1} c \left[\sum_{k=1}^c \frac{(T + T_3)^k}{k!} e^{-\Lambda(T+T_3)} + \frac{e^{-\Lambda(T+T_3)}}{\Lambda} \right]' \\ &= c\Lambda^{c-1} e^{-\Lambda(T+T_3)} \left[1 + \sum_{k=1}^c \frac{(T + T_3)^k}{(k-1)!} \right. \\ &\quad \left. \left(\frac{\Lambda T + \Lambda T_3}{k} - 1 \right) \right] \end{aligned} \quad (17)$$

$$\approx c\Lambda^{c-1} e^{-\Lambda(T+T_3)}$$

$$f_3'(T) = P_2' \cdot cT_2. \quad (18)$$

Clearly, $f_3'(T)$ remains consistently positive. Next, we judge whether $(f_1'(T) + f_2'(T))$ is positive or negative:

$$\begin{aligned} f_1'(T) + f_2'(T) &= e^{-\Lambda(T+T_3)} \\ &\left\{ c\Lambda^{c-1} + \left[\Lambda T + \Lambda T_3 + \sum_{k=1}^{c-1} \frac{(\Lambda T + \Lambda T_3)^k}{k!} - \Gamma \right] (1 + \Lambda T_2) \right\} \\ &> e^{-\Lambda(T+T_3)} \left\{ c\Lambda^{c-1} + \left[\sum_{k=1}^{c-1} \frac{(\Lambda T + \Lambda T_3)^k}{k!} - \Gamma \right] (1 + \Lambda T_2) \right\} \\ &> e^{-\Lambda(T+T_3)} \left\{ c\Lambda^{c-1} + \left[\frac{(\Lambda T + \Lambda T_3)^{c-1}}{(c-1)!} - \Gamma \right] (1 + \Lambda T_2) \right\} \\ &> \frac{\Lambda^{c-1} e^{-\Lambda(T+T_3)}}{(c-1)!} [c! + (1 - \Lambda T - \Lambda T_3)(1 + \Lambda T_2)(T + T_3)^{c-1}] \end{aligned} \quad (19)$$

where

$$\Gamma = \frac{(\Lambda T + \Lambda T_3)^c}{(c-1)!}. \quad (20)$$

Since $(T + T_3)$ is larger than c/Λ , which is significantly smaller than 1, $(f_1'(T) + f_2'(T))$ is greater than 0. Consequently, the derivative of the average latency $D(T)$ remains consistently positive, indicating that $D(T)$ of *StreamFrames* is *positively correlated with T*. This conclusion proves that adjusting T significantly impacts data delivery performance when aiming to reduce communication overhead.

VI. EXPERIMENTAL RESULTS

In this section, we first introduce the evaluation settings. Second, we validate the effectiveness of ShuttleBus and its components on communication overhead. We also compare ShuttleBus with benchmarks to demonstrate its benefits in data delivery under differentiated latency constraints. Last, we conduct an in-depth analysis of the packet assembling process and the involved parameters.

A. Evaluation Setting

1) *Experimental Environment*: Constructing an actual mIoT network deploying thousands of MTC entities is challenging. Hence, we design a practical approach based on some docker containers equipped with the QUIC protocol,² enhanced by the proposed ShuttleBus. In this approach, a container can function as a BS, an MTCG, or a group of MTCs, as illustrated in Fig. 9. To measure the latency accurately, all containers are deployed in a Dell workstation (CPU: Intel I7-10700 2.9 GHz, Memory: 64 G DDR4 3200 MHz). In the intra-clustering transmission phase, each MTC container builds multiple QUIC connections with an MTCG container by utilizing different ports. In this phase, each connection contains a single QUIC stream to transmit the raw data of the corresponding MTC, and these connections operate concurrently to simulate transmission from multiple independent MTCs. In the MTCG forwarding phase,

²The code of the enhanced QUIC protocol is modified based on the *quic-go* project (<https://github.com/lucas-clemente/quic-go>).

TABLE I
EXPERIMENTAL NETWORK TRACES

Trace1	MTCD number	681	147	581	125	356	294	562	428	811	237	795	528	847	287	
	duration (ms)	7000	9000	8000	5000	5000	7000	9000	9000	5000	6000	6000	8000	7000	9000	
Trace2	MTCD number	125	238	534	180	613	810	955	741	484	632	397	541	218	720	611
	duration (ms)	6000	8000	5000	7000	9000	8000	7000	7000	7000	5000	5000	5000	5000	8000	8000
Trace3	MTCD number	480	492	326	858	859	220	452	281	373	429	548	474	529	192	498
	duration (ms)	5000	6000	6000	9000	5000	9000	7000	7000	5000	8000	5000	8000	8000	5000	7000
Trace4	MTCD number	427	533	327	127	891	379	403	195	114	654	594	425	229	846	702
	duration (ms)	7000	9000	9000	8000	5000	5000	6000	6000	9000	8000	6000	5000	6000	5000	6000

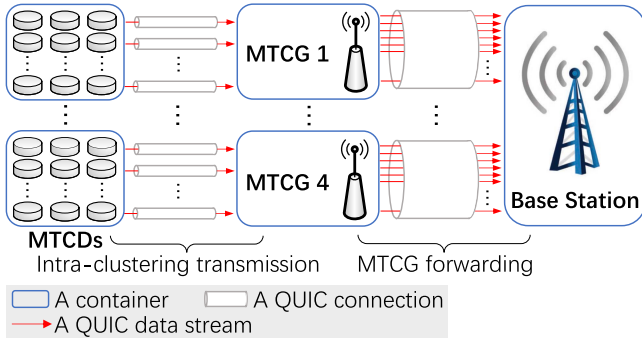


Fig. 9. An illustration of our simulated network.

each MTCG container builds a single QUIC connection with the unique central BS container, and hundreds of streams run simultaneously within this connection. In this way, the approach effectively simulates the two-hop communication in an mIoT network.

2) *Experimental Setting*: Referring to related works [4], [51], [52], we configure certain network parameters as follows. To clearly distinguish the transmission priorities of MTCDs clearly, we categorize them into three types with different priority levels: high-priority (HP), regular-priority (RP), and low-priority (LP) (the number of MTCD priority levels can be adjusted based on specific mIoT scenarios). The latency constraints and packet generation intervals for HP, RP, and LP MTCDs are set at $1ms$ and $2ms$, $10ms$ and $10ms$, and $50ms$ and $20ms$, respectively. Among all MTCDs, the distribution of HP, RP, and LP MTCDs is 5%, 45%, and 50%.

To generate various network load conditions, we create a BS container, four MTCG containers, and four MTCD containers that simulate a total MTCD count ranging from 400 to 4,000. We set the average payload length of packets sent from MTCDs at 50 bytes, thus the network load degree of an MTCG is directly proportional to the number of connected MTCDs. Except for fixed network loads, we generate four random network traces with varying numbers of MTCDs and durations, as listed in Table I. In each trace, the number of MTCDs varies from 400 to 4,000, and the duration ranges from 5,000 to 10,000 milliseconds. Whenever the number of MTCDs changes, we randomly select corresponding numbers of MTCDs with different priority levels and update their latency constraints using the proposed *StreamPropertyFrame* in the network.

To simulate the wireless links between MTCGs and MTCDs, we configure the data transmission rate of the links at $3 MB/s$ with a 0.1% random packet loss rate [53]. In these links, the transmission latency of a packet is about $300 \mu s$, which has been taken into account when calculating the total latency. The links between the MTCGs and the BS are set as wireless (with a 0.1% random packet loss rate), and their data transmission rates are set at $100 MB/s$. In typical mIoT scenarios, the total distance between MTCDs and the BS spans hundreds of meters to several kilometers. Consequently, the total propagation time is only several microseconds, which is not considered a bottleneck.

B. Performance on Communication Overhead

We collect two metrics of communication overhead in MTCG containers: the average forwarding packet rate and the average forwarding data rate incurred by transmitting the same amount of user data. The data rate encompasses both packet headers and payloads. We compare ShuttleBus with conventional forwarding approaches using TCP and QUIC connections (denoted as *TCP-Forwarding* and *QUIC-Forwarding*, respectively) without data merging. Besides, we conduct distillation experiments to analyze the impact of ShuttleBus components. Specifically, we document the results of the pure SMDM approach (denoted as *SMDM-only*), the SMDM approach with RFP (denoted as *SMDM-RFP*), and the SMDM approach with LONS (denoted as *SMDM-LONS*). In the following experiments, the DRL models of ShuttleBus and *SMDM-RFP* are trained offline first by using the simulated data before conducting online inference tasks.

Tables II and III summarize the performance of these approaches at MTCGs with fixed and dynamic network loads, respectively. We observe that: (i) *TCP-forwarding* and *QUIC-forwarding* send a similar number of packets, but *QUIC-forwarding* consumes more bandwidth resources due to the short frame headers of QUIC *StreamFrames*. (ii) Compared with the conventional forwarding schemes, SMDM-based approaches (including ShuttleBus) show apparent advantages in the communication overhead, indicating that merging multiple short packets effectively conserves network resources. (iii) The RFP mechanism significantly reduces the packet rate and data rate, as the average multiplexed frame number per packet dramatically increases. (iv) The LONS mechanism decreases communication overhead by effectively scheduling data transmission under dynamic network loads. (v) By integrating these mechanisms, ShuttleBus derives the best performance in communication overhead reduction among all schemes under most network

TABLE II
COMMUNICATION OVERHEAD DURING THE FORWARDING WITH FIXED NETWORK LOADS

Approaches	1000 MTCDs		2000 MTCDs		3000 MTCDs		4000 MTCDs	
	packet rate (/s)	data rate (MB/s)	packet rate (/s)	data rate (MB/s)	packet rate (/s)	data rate (MB/s)	packet rate (/s)	data rate (MB/s)
TCP-Forwarding	23275.8	3.076	48217.5	6.036	69586.4	8.975	92178.6	11.711
QUIC-Forwarding	23386.6	3.126	47719.0	6.203	69417.9	9.183	91941.5	12.011
Cascading Timer	6002.5	2.439	7353.4	4.748	8213.1	7.339	8481.6	9.448
SMDM-only	19319.8	2.912	36612.4	5.801	47571.2	8.392	53060.8	10.783
SMDM-RFP	1735.1	2.294	3528.7	4.520	5309.6	6.834	6843.3	9.109
SMDM-LONS	20014.5	2.938	36238.4	5.786	47711.1	8.398	52190.5	10.762
ShuttleBus	1817.2	2.347	3573.2	4.576	5231.1	6.830	6744.2	9.102

TABLE III
COMMUNICATION OVERHEAD DURING THE FORWARDING WITH DYNAMIC NETWORK LOADS

Approaches	trace1		trace2		trace3		trace4	
	packet rate (/s)	data rate (MB/s)	packet rate (/s)	data rate (MB/s)	packet rate (/s)	data rate (MB/s)	packet rate (/s)	data rate (MB/s)
TCP-Forwarding	37841.9	5.012	47715.8	6.186	41285.2	5.401	35681.8	4.639
QUIC-Forwarding	38025.1	5.074	47404.8	6.273	41368.0	5.481	36058.4	4.768
Cascading Timer	7540.3	4.013	8152.7	5.102	7701.1	4.095	7281.3	3.740
SMDM-only	23148.5	4.489	28321.9	5.512	23770.6	4.619	21774.7	4.196
SMDM-RFP	6932.3	3.896	6749.4	4.841	6960.6	4.047	6761.6	3.593
SMDM-LONS	23007.4	4.419	25269.6	5.472	23421.9	4.538	20337.8	4.073
ShuttleBus	6808.3	3.824	6685.9	4.722	6862.7	3.934	6712.7	3.580

conditions, reducing the packet rate by over 90% and the data rate by about 30% to 45%.

C. Performance on Data Delivery (Latency)

In this subsection, we focus on determining whether the data violate latency constraints before delivery. To evaluate this, we define two metrics: the proportion of waiting timeout *StreamFrames* (PWTS) and the proportion of delivery timeout *StreamFrames* (PDTS). PWTS and PDTS are the proportions of *StreamFrames* that exceed the latency constraints during backlog and before delivering to the BS, respectively. We compare the ShuttleBus scheme with the conventional *QUIC-forwarding* approach and conduct the same distillation experiments as Section VI-B to evaluate the impact of the proposed components on data delivery.

In our testing, the values of PDTS and PWTS are minimal when the network load is low. Hence, in experiments with fixed network loads, we assess the data delivery performance in mIoT networks, with the total number of MTCDs ranging from 2,800 to 4,000. As plotted in Fig. 10, we observe that: (i) *QUIC-Forwarding* exhibits the poorest performance under all network conditions due to inefficient resource utilization without data merging. Besides, the performance difference between *QUIC-Forwarding* and other schemes becomes more pronounced with increased network load. (ii) Upon adopting the RFP or LONS mechanisms, both metrics show marked declines. We can conclude that the RFP and LONS mechanisms contribute to satisfying latency constraints in mIoT scenarios. With these

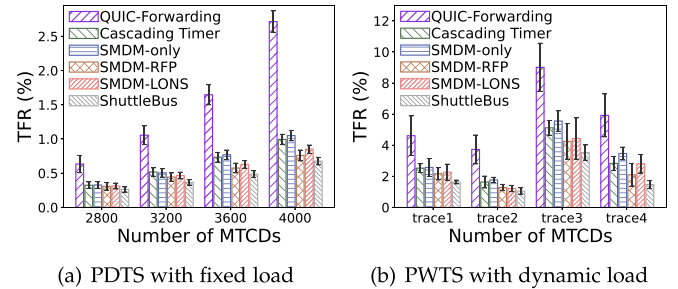


Fig. 10. Data delivery performance of forwarding approaches under different network load conditions.

components, ShuttleBus reduces the proportions of timeout *StreamFrames* by approximately 50% to 85%. (iii) Compared with *SMDM-LONS*, although the PDTS of ShuttleBus may be marginally larger when the network load is light, the PWTS of ShuttleBus is considerably smaller under all conditions. This outcome is attributable to the RFP mechanism's ability to reduce congestion probability and data backlogged time, as demonstrated in Section V-A.

D. In-Depth Analysis of ShuttleBus

1) *Probabilities of Three Possible Cases*: To investigate the packet assembly process of ShuttleBus, we document the alterations in the probabilities of three potential cases after implementing the RFP mechanism, as discussed in Sections VI. Fig. 11(a) and (b) display the average results at the MTCGs under

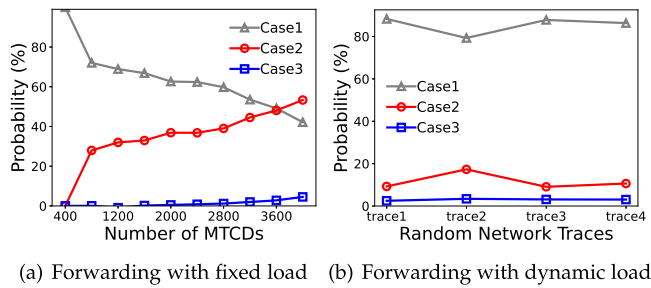
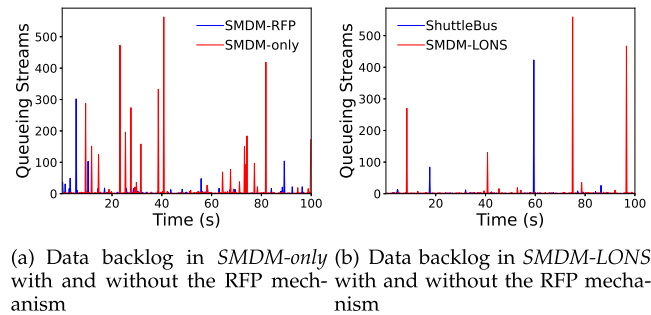
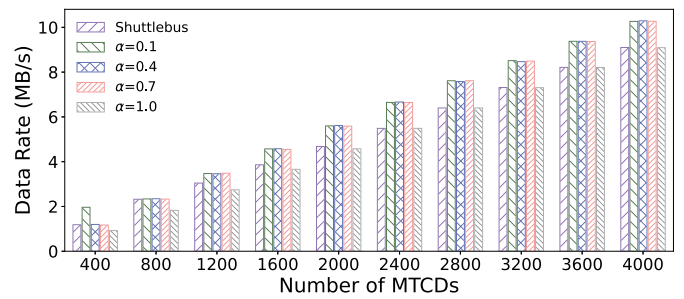


Fig. 11. Probabilities of the cases under different network load conditions.

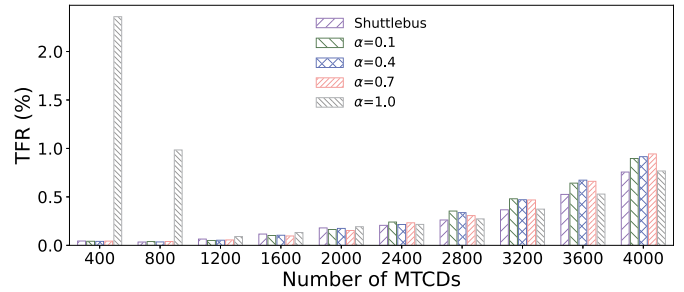
Fig. 12. Number of backlogged *StreamFrames* at the MTCGs.

fixed and dynamic network loads, respectively. In Fig. 11(a), the constraint on tolerance assembling time leads to the predominance of *Case1* under most conditions, preventing the sender from blindly pursuing a reduction in communication overhead at the expense of increased latency. As the fixed network load rises, the probabilities of *Case2* and *Case3* both increase, implying a larger average number of multiplexed *StreamFrames* within each QUIC packet. In Fig. 11(b), the probabilities of *Case1* are substantially higher than those of the other two cases across all random network traces. The reason is that Shuttlebus assembles a greater number of packets under low network load conditions to ensure adherence to latency constraints. These findings confirm that the packet assembling process of ShuttleBus effectively balances communication overhead and latency.

2) *Congestion Conditions*: To access congestion at the MTCGs, we sample the number of active streams in the queues every 200 ms under the heaviest network load (i.e., 4,000 MTCDs). We illustrate the collected data when the RFP mechanism is incorporated into the *SMDM-only* and *SMDM-LONS* schemes in Fig. 12(a) and (b), respectively. As shown, in all schemes, sudden congestion is not entirely avoided due to the limited processing capacity of MTCGs. However, the congestion in ShuttleBus is significantly alleviated in frequency after integrating the RFP mechanism into both the *SMDM-only* and *SMDM-LONS* schemes. This is because the proposed RFP mechanism diminishes the congestion probability during the packet assembling process, as analyzed in Section V-A. Moreover, in comparison to the backlogged conditions in *SMDM-only*, *SMDM-LONS* considerably reduces the potential congestion occurring at the MTCGs. This outcome proves that



(a) Data rate

(b) Timeout *StreamFrame* ratioFig. 13. Data rate and timeout *StreamFrame* ratio (TFR) of an MTCG using different values of α with fixed network loads.

the proposed LONS mechanism also contributes to efficient and timely data forwarding through the use of latency-oriented scheduling.

3) *Impact of α in Equation (3)*: To evaluate the impact of α on the merging efficiency (i.e., communication overhead reduction) and latency of *StreamFrames*, we conduct experiments where α ranges from 0.1 to 1.0. Fig. 13 plots the average packet rate and timeout *StreamFrame* ratio (i.e., the sum of PDTS and PWTS) at each MTCG, with a fixed total number of connected MTCDs ranging from 400 to 4,000. In the figure, a larger α value results in higher merging efficiency owing to an extended tolerance assembling time; however, it also increases the risk of timeout when the network load is low. The reason is that most packets wait for an entire tolerance assembling time before being sent under a low network load, causing the data delivery performance to be easily affected by unpredictable network fluctuations. Conversely, a small α value does not guarantee meeting latency constraints for data under a heavy network load, as this leads to inefficient packet merging at the MTCGs.

By adopting the NLAW algorithm, the proposed ShuttleBus selects the optimal α value according to real-time network load conditions. As shown in Fig. 13, ShuttleBus with the NLAW algorithm achieves superior data delivery performance compared to using fixed α values. Furthermore, although ShuttleBus consumes slightly more communication resources when the network load is low, it yields the lowest communication overhead under other network load conditions.

Additionally, we compare the ShuttleBus scheme with various fixed values of α at the MTCGs under dynamic network loads,

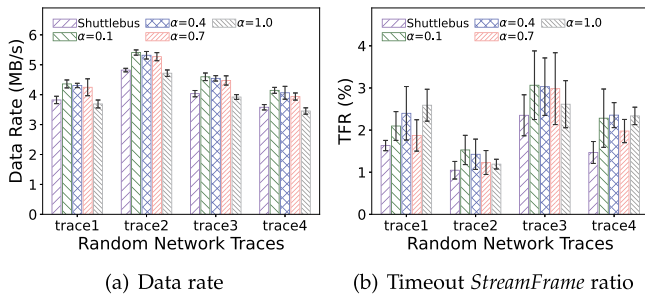


Fig. 14. Data rate and timeout *StreamFrame* ratio (TFR) of an MTCG using different values of α with dynamic network loads.

exhibiting the results in Fig. 14. As shown, fixed α values cannot simultaneously optimize communication overhead and timely data delivery metrics under dynamic network loads. In contrast, ShuttleBus obtains the best performance for both metrics across all random network traces. This is due to its ability to adapt α values according to real-time network load conditions, seeking to minimize communication overhead while adhering to latency constraints. In summary, the proposed ShuttleBus effectively balances the trade-off between communication overhead and data delivery performance under all network load conditions.

VII. CONCLUSION

In this paper, we have proposed a QUIC-based dense packet assembling scheme, ShuttleBus, to address the short packet forwarding issue at MTCGs for dynamic clustering-based mIoT. With ShuttleBus, the QUIC protocol can be enhanced to reduce the communication overhead and satisfy differentiated latency constraints when aggregating and forwarding the transmission of massive parallel connections. The advantages and practicality of ShuttleBus are: (i) it preserves data integrity of QUIC streams during packet reassembling and massive end-to-end transmissions, while providing excellent scalability for adjusting the stream count; (ii) its improvement of network resource utilization significantly increases the afforded MTC number under a central BS and supports more dense communications for an mIoT scenario; (iii) since the underlying QUIC protocol operates in the user space, ShuttleBus is compatible with most real-world middlebox implementations. The QUIC-based ShuttleBus can be deployed in emerging massive MTC scenarios to improve network performance. For future work, we will investigate more flexible data scheduling approaches, such as data replication and data acknowledgment for QUIC streams traversing multiple wireless paths, to further enhance the transmission reliability.

REFERENCES

- [1] E. Sisinni, A. Saifullah, S. Han, U. Jennehag, and M. Gidlund, "Industrial Internet of Things: Challenges, opportunities, and directions," *IEEE Trans. Ind. Inform.*, vol. 14, no. 11, pp. 4724–4734, Nov. 2018.
- [2] J. Bai, H. Song, Y. Yi, and L. Liu, "Multiagent reinforcement learning meets random access in massive cellular Internet of Things," *IEEE Internet Things J.*, vol. 8, no. 24, pp. 17 417–17 428, Dec. 2021.
- [3] M. Giordani, M. Polese, M. Mezzavilla, S. Rangan, and M. Zorzi, "Toward 6G networks: Use cases and technologies," *IEEE Commun. Mag.*, vol. 58, no. 3, pp. 55–61, Mar. 2020.

- [4] 6G: The next horizon white paper, Huawei, 2021. [Online]. Available: <https://www.huawei.com/en/technology-insights/future-technologies/6g-white-paper>
- [5] U. Tefek and T. J. Lim, "Full-duplex relaying in machine-type communications with a multi-antenna base station," *IEEE Trans. Wireless Commun.*, vol. 17, no. 9, pp. 5804–5817, Sep. 2018.
- [6] S. Zhang, P. He, K. Suto, P. Yang, L. Zhao, and X. Shen, "Cooperative edge caching in user-centric clustered mobile networks," *IEEE Trans. Mobile Comput.*, vol. 17, no. 8, pp. 1791–1805, Aug. 2018.
- [7] T. Wang, Y. Wang, Z. Yang, and J. Cheng, "Group-based random access and data transmission scheme for massive MTC networks," *IEEE Trans. Commun.*, vol. 69, no. 12, pp. 8287–8303, Dec. 2021.
- [8] 3GPP, "Service requirements for machine-type communications," 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 22.368, version 17.0.0, Dec. 2014. [Online]. Available: <https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=645>
- [9] X. Shen, G. Jie, W. Wu, M. Li, C. Zhou, and W. Zhuang, "Holistic network virtualization and pervasive network intelligence for 6G," *IEEE Commun. Surveys Tuts.*, vol. 24, no. 1, pp. 1–30, First Quarter, 2022.
- [10] J. Postel, "Transmission control protocol," RFC 793, Sep. 1981. [Online]. Available: <https://www.rfc-editor.org/info/rfc793>
- [11] D. Borman, B. Braden, V. Jacobson, and R. Scheffegger, "TCP extensions for high performance," RFC 7323, Sep. 2014. [Online]. Available: <https://www.rfc-editor.org/info/rfc7323>
- [12] A. S. Akyurek and T. S. Rosing, "Optimal packet aggregation scheduling in wireless networks," *IEEE Trans. Mobile Comput.*, vol. 17, no. 12, pp. 2835–2853, Dec. 2018.
- [13] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "TAG: A tiny aggregation service for ad-hoc sensor networks," in *Proc. 5th Symp. Operating Syst. Des. Implementation*, Boston, MA, USA, 2002, pp. 131–146.
- [14] O. Younis and S. Fahmy, "HEED: A hybrid, energy-efficient, distributed clustering approach for ad hoc sensor networks," *IEEE Trans. Mobile Comput.*, vol. 3, no. 4, pp. 366–379, Fourth Quarter, 2004.
- [15] F. Ren, J. Zhang, Y. Wu, T. He, C. Chen, and C. Lin, "Attribute-aware data aggregation using potential-based dynamic routing in wireless sensor networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 5, pp. 881–892, May 2013.
- [16] R. Stewart, "Stream control transmission protocol," RFC 4960, Sep. 2007. [Online]. Available: <https://www.rfc-editor.org/info/rfc4960>
- [17] R. Stewart, M. Tüxen, and M. Proshin, "Stream control transmission protocol: Errata and issues in RFC 4960," RFC 8540, Feb. 2019. [Online]. Available: <https://www.rfc-editor.org/info/rfc8540>
- [18] E. Volodina and E. P. Rathgeb, "Flow control in the context of the multiplexed transport protocol QUIC," in *Proc. IEEE 45th Conf. Local Comput. Netw.*, Sydney, Australia, 2020, pp. 473–478.
- [19] A. Langley et al., "The QUIC transport protocol: Design and internet-scale deployment," in *Proc. Conf. ACM Special Int. Group Data Commun.*, Los Angeles, CA, USA, 2017, pp. 183–196.
- [20] M. Bishop, "Http/3," RFC 9114, Jun. 2022. [Online]. Available: <https://www.rfc-editor.org/info/rfc9114>
- [21] Q. Li, J. Chen, M. Cheffena, and X. Shen, "Channel-aware latency tail taming in Industrial IoT," *IEEE Trans. Wireless Commun.*, vol. 22, no. 9, pp. 6107–6123, Sep. 2023.
- [22] S. Yan, Q. Ye, and W. Zhuang, "Learning-based transmission protocol customization for VoD streaming in cybertwin-enabled next-generation core networks," *IEEE Internet Things J.*, vol. 8, no. 22, pp. 16 326–16 336, Nov. 2021.
- [23] 5G for connected industries and automation, 5G Alliance Connected Ind. Autom., white paper, Frankfurt, Germany, Feb. 2019.
- [24] T. Zhang, T. Gong, S. Han, Q. Deng, and X. S. Hu, "Fully distributed packet scheduling framework for handling disturbances in lossy real-time wireless networks," *IEEE Trans. Mobile Comput.*, vol. 20, no. 2, pp. 502–518, Feb. 2021.
- [25] Q. Ye, W. Shi, K. Qu, H. He, W. Zhuang, and X. Shen, "Joint RAN slicing and computation offloading for autonomous vehicular networks: A learning-assisted hierarchical approach," *IEEE Open J. Veh. Technol.*, vol. 2, pp. 272–288, Jun. 2021.
- [26] J. Iyengar and M. Thomson, "QUIC: A UDP-based multiplexed and secure transport," RFC 9000, May 2021. [Online]. Available: <https://www.rfc-editor.org/info/rfc9000>
- [27] Q. Xiang, H. Zhang, J. Xu, X. Liu, and L. J. Rittle, "When in-network processing meets time: Complexity and effects of joint optimization in wireless sensor networks," *IEEE Trans. Mobile Comput.*, vol. 10, no. 10, pp. 1488–1502, Oct. 2011.

- [28] R. Rajagopalan and P. K. Varshney, "Data-aggregation techniques in sensor networks: A survey," *IEEE Commun. Surveys Tut.*, vol. 8, no. 4, pp. 48–63, Fourth Quarter, 2006.
- [29] E. Fasolo, M. Rossi, J. Widmer, and M. Zorzi, "In-network aggregation techniques for wireless sensor networks: A survey," *IEEE Wireless Commun.*, vol. 14, no. 2, pp. 70–87, Apr. 2007.
- [30] K. Fan, S. Liu, and P. Sinha, "Structure-free data aggregation in sensor networks," *IEEE Trans. Mobile Comput.*, vol. 6, no. 8, pp. 929–942, Aug. 2007.
- [31] I. Solis and K. Obraczka, "In-network aggregation trade-offs for data collection in wireless sensor networks," *Int. J. Sensor Netw.*, vol. 1, no. 3/4, pp. 200–212, Jan. 2006.
- [32] K. Fan, S. Liu, and P. Sinha, "On the potential of structure-free data aggregation in sensor networks," in *Proc. IEEE 25th Int. Conf. Comput. Commun.*, Barcelona, Spain, 2006, pp. 1–12.
- [33] W. B. Heinzelman, A. P. Chandrakasan, and H. Balakrishnan, "An application-specific protocol architecture for wireless microsensor networks," *IEEE Trans. Wireless Commun.*, vol. 1, no. 4, pp. 660–670, Oct. 2002.
- [34] J. Guo, S. Durrani, X. Zhou, and H. Yanikomeroglu, "Massive machine type communication with data aggregation and resource scheduling," *IEEE Trans. Commun.*, vol. 65, no. 9, pp. 4012–4026, Sep. 2017.
- [35] O. L. A. López, H. Alves, P. H. J. Nardelli, and M. Latva-aho, "Aggregation and resource scheduling in machine-type communication networks: A stochastic geometry approach," *IEEE Trans. Wireless Commun.*, vol. 17, no. 7, pp. 4750–4765, Jul. 2018.
- [36] D. M. Kim, R. B. Sorensen, K. Mahmood, O. N. Osterbo, A. Zanella, and P. Popovski, "Data aggregation and packet bundling of uplink small packets for monitoring applications in LTE," *IEEE Netw.*, vol. 31, no. 6, pp. 32–38, Nov./Dec. 2017.
- [37] R. Abbas, M. Shirvanimoghaddam, Y. Li, and B. Vucetic, "Random access for M2M communications with QoS guarantee," *IEEE Trans. Commun.*, vol. 65, no. 7, pp. 2889–2903, Jul. 2017.
- [38] A.-T. H. Bui, C. T. Nguyen, T. C. Thang, and A. T. Pham, "A comprehensive distributed queue-based random access framework for mMTC in LTE/LTE-A networks with mixed-type traffic," *IEEE Trans. Veh. Technol.*, vol. 68, no. 12, pp. 12 107–12 120, Dec. 2019.
- [39] W. Cao, A. Dytso, G. Feng, H. V. Poor, and Z. Chen, "Differentiated service-aware group paging for massive machine-type communication," *IEEE Trans. Commun.*, vol. 66, no. 11, pp. 5444–5456, Nov. 2018.
- [40] X. Chen, D. W. K. Ng, W. Yu, E. G. Larsson, N. Al-Dhahir, and R. Schober, "Massive access for 5G and beyond," *IEEE J. Sel. Areas Commun.*, vol. 39, no. 3, pp. 615–637, Mar. 2021.
- [41] W. Saad, M. Bennis, and M. Chen, "A vision of 6G wireless systems: Applications, trends, technologies, and open research problems," *IEEE Netw.*, vol. 34, no. 3, pp. 134–142, May/June. 2020.
- [42] C. Bockelmann et al., "Massive machine-type communications in 5G: Physical and MAC-layer solutions," *IEEE Commun. Mag.*, vol. 54, no. 9, pp. 59–65, Sep. 2016.
- [43] A. Joseph, T. Li, Z. He, Y. Cui, and L. Zhang, "A comparison between SCTP and QUIC," Internet Eng. Task Force, Tech. Rep. draft-joseph-quick-comparison-sctp-00, Mar. 2018, p. 24. [Online]. Available: <https://datatracker.ietf.org/doc/draft-joseph-quick-comparison-sctp-00/>
- [44] V. Mnih et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015.
- [45] R. Bellman, "A Markovian decision process," *J. Math. Mechanics*, vol. 6, pp. 679–684, 1957.
- [46] W. Zhang et al., "Optimizing federated learning in distributed Industrial IoT: A multi-agent approach," *IEEE J. Sel. Areas Commun.*, vol. 39, no. 12, pp. 3688–3703, Dec. 2021.
- [47] S. Fujimoto, H. V. Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *Proc. 35th Int. Conf. Mach. Learn.*, Stockholm, Sweden, 2018, pp. 1582–1591.
- [48] N. T. J. Bailey, "On queueing processes with bulk service," *J. Roy. Statist. Soc. Ser. Methodol.*, vol. 16, no. 1, pp. 80–87, 1954.
- [49] S. Sasikala and K. Indhira, "Bulk service queueing models - A survey," *Int. J. Pure Appl. Math.*, vol. 106, no. 6, pp. 43–56, Jan. 2016.
- [50] D. Claeys, B. Steyaert, J. Walraevens, K. Laevens, and H. Bruneel, "Analysis of a versatile batch-service queueing model with correlation in the arrival process," *Int. J. Pure Appl. Math.*, vol. 70, no. 4, pp. 300–316, Apr. 2013.
- [51] J. Gao, W. Zhuang, M. Li, X. Shen, and X. Li, "MAC for machine-type communications in industrial IoT - Part I: Protocol design and analysis," *IEEE Internet Things J.*, vol. 8, no. 12, pp. 9945–9957, Jun. 2021.

- [52] J. Gao, M. Li, W. Zhuang, X. Shen, and X. Li, "Mac for machine type communications in Industrial IoT - Part II: Scheduling and numerical results," *IEEE Internet Things J.*, vol. 8, no. 12, pp. 9958–9969, Jun. 2021.
- [53] H. Shariatmadari et al., "Machine-type communications: Current status and future perspectives toward 5G systems," *IEEE Commun. Mag.*, vol. 53, no. 9, pp. 10–17, Sep. 2015.



Bo He (Member, IEEE) received the PhD degree from the Beijing University of Posts and Telecommunications, China, in 2023. He is currently a postdoctoral researcher with the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications. From 2021 to 2022, he was a visiting PhD student with the University of Waterloo, Canada. His research interests include 5 G/6 G networks, multipath networks, collective communication, transmission control, and deep reinforcement learning.



Jingyu Wang (Senior Member, IEEE) received the PhD degree from the Beijing University of Posts and Telecommunications, in 2008. He is currently a professor with the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications. He is a senior member of the China Communication Society and was selected for the Beijing Young Talents Program. He has published more than 100 papers in international journals or conferences, including the *IEEE Communications Magazine*, *IEEE Transactions on Services Computing*, *IEEE Transactions on Multimedia*, CVPR, ACL, ICDE, AAAI, and so on. His research interests span broad aspects of intelligent networks, edge/cloud computing, machine learning, AI/ops, IoV/IoT, SDN/NFV, knowledge-defined network, and intent-based networking.



Qi Qi (Senior Member, IEEE) received the PhD degree from the Beijing University of Posts and Telecommunications, in 2010. Currently, she is a professor with the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications. She has published more than 30 papers in international journals or conferences, and obtained two National Natural Science Foundations of China. Her research interests include edge intelligence, the Internet of Things, multimedia services, deep reinforcement learning, and distributed machine learning.



Qiang Ye (Senior Member, IEEE) received the PhD degree in electrical and computer engineering from the University of Waterloo, ON, Canada, in 2016. Since 2023, he has been an assistant professor with the Department of Electrical and Software Engineering, University of Calgary, AB, Canada. Before joining UCalgary, he worked as an assistant professor with the Department of Computer Science, Memorial University of Newfoundland, NL, Canada from 2021 to 2023 and with the Department of Electrical and Computer Engineering and Technology, Minnesota State University, USA, from 2019 to 2021, respectively. He was with the Department of Electrical and Computer Engineering, University of Waterloo as a postdoctoral fellow and then a research associate from 2016 to 2019. He has published around 70 research articles on top-ranked Journals and Conference Proceedings. He is/was the general, publication, program co-chairs for different international conferences and workshops, e.g., IEEE ICC'23, CANAI'23, IEEE VTC'22, IEEE INFOCOM'22, and IEEE IPCCC'21. He serves/served as associate editors of *IEEE Transactions on Vehicular Technology*, *IEEE Transactions on Cognitive Communications and Networking*, *IEEE Open Journal of the Communications Society*, *Peer-to-Peer Networking and Applications*, *ACM/Wireless Networks*, and *International Journal of Distributed Sensor Networks*. He also serves as the IEEE Vehicular Technology Society (VTS) Regions 1-7 Chapters coordinator (2022–2023).



Qihao Li (Member, IEEE) received the MSc degree in information and communication technology from the University of Agder, Norway, in 2013, and the PhD degree from the Department of Electrical and Computer Engineering, University of Oslo, Norway, in 2019. In 2016, he was a visiting researcher with the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON, Canada. In 2020, he was a postdoctoral fellow with the Department of Electrical and Computer Engineering, University of Waterloo. He is currently an associate professor with the College of Communication Engineering, Jilin University, China.

His current research focuses on industrial Internet, digital twin, optimal control and optimization, wireless network security, and localization. He served as the a member of Technical Program Committee for IEEE Globecom'19-22, IEEE ICC'19-22, IEEE CIC ICC'17-23, EuCAP'2019, and BDEC-SmartCity'18.



Jianxin Liao received the PhD degree from the University of Electronics Science and Technology of China, in 1996. He is currently the dean of the Network Intelligence Research Center and the full professor of the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications. He has published hundreds of research papers and several books. He has won a number of prizes in China for his research achievements, which include the Premier's Award of Distinguished Young Scientists from the National

Natural Science Foundation of China in 2005, and the Specially-invited professor of the "Yangtse River Scholar Award Program" by the Ministry of Education in 2009. His main research interests include cloud computing, mobile intelligent network, service network intelligence, networking architectures and protocols, and multimedia communication.



Xuemin Shen (Fellow, IEEE) received the PhD degree in electrical engineering from Rutgers University, New Brunswick, NJ, USA, in 1990. He is currently a university professor with the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON, Canada. His research focuses on resource management in interconnected wireless/wired networks, wireless network security, social networks, smart grid, and vehicular ad hoc and sensor networks. He is a registered professional engineer of Ontario, Canada, fellow of the Engineering

Institute of Canada, fellow of the Canadian Academy of Engineering, fellow of the Royal Society of Canada, foreign member of the Chinese Academy of Engineering, and distinguished lecturer of IEEE Vehicular Technology Society and Communications Society. He was the recipient of Canadian Award for Telecommunications Research from the Canadian Society of Information Theory (CSIT) in 2021, R.A. Fessenden Award in 2019 from IEEE, Canada, Award of Merit from the Federation of Chinese Canadian Professionals (Ontario) in 2019, James Evans Avant Garde Award in 2018 from the IEEE Vehicular Technology Society, Joseph LoCicero Award in 2015 and Education Award in 2017 from the IEEE Communications Society, and Technical Recognition Award from Wireless Communications Technical Committee (2019) and AHSN Technical Committee (2013). He was also the recipient of Excellent Graduate Supervision Award in 2006 from the University of Waterloo and Premier's Research Excellence Award (PREA) in 2003 from the Province of Ontario, Canada. He was the technical program committee chair/co-chair of IEEE Globecom'16, IEEE Infocom'14, IEEE VTC'10 Fall, IEEE Globecom'07, and the chair of IEEE Communications Society Technical Committee on Wireless Communications. He is the president of IEEE Communications Society. He was the vice president of Technical and Educational Activities, vice president for Publications, member-at-large on the Board of Governors, chair of the Distinguished Lecturer Selection Committee, a member of IEEE Fellow Selection Committee of the ComSoc. He was the editor-in-chief of the *IEEE Internet of Things Journal*, *IEEE Network*, and *IET Communications*.