Design and Implementation of Holistic Scheduling and Efficient Storage for FlexRay

Yu Hua, *Senior Member, IEEE*, Xue Liu, *Member, IEEE*, Wenbo He, *Member, IEEE*, and Dan Feng, *Member, IEEE*

Abstract—FlexRay is a new industry standard for next-generation communication in automotives. Though there are a few recent researches on performance analysis of FlexRay, two important aspects of the FlexRay design have been overlooked. The first is a holistic integrated scheduling scheme that can handle both static and dynamic segments in a FlexRay network. The second is cost-effective and scalable performance. To address these aspects, we propose a novel holistic scheduling scheme, called HOSA, which can provide scalable performance by using flexible and ease-of-use dual channel communication in FlexRay. HOSA is built upon a novel slot pilfering technique to schedule and optimize the available slots in both static and dynamic segments. Moreover, to achieve efficient implementation, we propose approximate computation, which can efficiently support cost-effective and holistic scheduling by judiciously obtaining the tradeoff between computation complexity and available pilfered slots. HOSA hence offers the salient feature of improving bandwidth utilization. Moreover, to deliver high performance and bridge the gap between real-time communications in networks and storage management in end devices, we implement a deduplication-aware scheme in a real SSD prototype that can reduce space overhead and extend SSD lifespan (by significantly reducing duplicate write operations). The deduplication scheme meets the needs of message updates in FlexRay. Extensive experiments based on synthetic test cases and real-world case studies demonstrate the efficiency and efficacy of HOSA.

Index Terms—Real-time scheduling, storage system, data deduplication

1 Introduction

Modern automobiles involve large amounts of sensors, actuators and Electronic Control Units (ECU) working together. This highly sophisticated interaction heavily relies on a communication system that connects different parts in an efficient manner [1], [2]. FlexRay [3] is an automotive network communications infrastructure developed by the FlexRay Consortium. The FlexRay consortium includes major car manufacturers such as BMW, Daimler-Benz, General Motors, Freescale, NXP, Bosch, and Volkswagen/Audi.

FlexRay has become one of the standards in the automotive industry. It provides a communication infrastructure for future generation high-speed X-by-wire applications in vehicles. These applications are mostly real-time and safety-critical [2]. FlexRay hence aims to provide hard real-time capabilities through cycle-based and time-triggered communications. The FlexRay standard is being deployed in the major line of new vehicles. For example, the 2007 BMW X5 is the first standard-production vehicle in the world to use this leading-edge technology [4]. To support autonomous vehicles, FlexRay controls the optical bus system of the Adaptive Drive chassis control system, and is used in controlling the stabilisers and electromagnetic valves of the dampers. This enables adaptive drive to eliminate the effect of body roll on the vehicle. It is envisioned in the near future that more and more functionalities in the autonomous vehicles will make use of the FlexRay bus system.

FlexRay provides two channels with a high bandwidth of 10 Mb/s each and offers multiple benefits compared with previous protocols, say Controller Area Network (CAN) [5], across a wide range of automotive applications. These benefits include high speed, fault tolerance, and a deterministic cycle-based message transport, along with a synchronized, common time base to all nodes in the system.

FlexRay is an open standard. It aims to provide scalable, deterministic and high performance communication for automotive applications. However, to be practical in automotive products and obtain significant performance improvements, we present three main observations below that meanwhile gain useful insights and motivate our design.

Observation 1: The Isolation of Scheduling Static and Dynamic Segments

FlexRay is a real-time system to schedule time-triggered and event-triggered messages. FlexRay supports the transmission of periodic messages in static segments (SS) and priority-based scheduling of event-triggered messages in dynamic segments (DS). Periodic messages are transmitted in the unique static slots of SS according to time division multiple access (TDMA). The operation of the FlexRay SS is similar to the time-triggered protocol (TTP) [6]. Moreover,

- *Y. Hua and D. Feng are with Wuhan National Laboratory for Optoelectronics, School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, China. E-mail: {csyhua, dfeng}@hust.edu.cn.*
- *X. Liu and W. He are with the School of Computer Science, McGill University, Montreal, Canada. E-mail: {xueliu,wenbohe}@cs.mcgill.ca.*

Manuscript received 6 May 2013; revised 22 July 2013; accepted 2 Aug. 2013. Date of publication 14 Aug. 2013; date of current version 17 Sept. 2014. Recommended for acceptance by X. Cheng.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPDS.2013.205

aperiodic messages are sent in the dynamic slots of DS that is similar to ByteFlight [7] and employs a flexible TDMA (FTDMA) approach. In both cases, the timely message delivery depends on the message schedule that is statically configured before the network starts to operate. The scheduling computation involves assigning the static slots for the periodic messages as well as the priority based dynamic slots assignment for the aperiodic messages. A message set is schedulable if a given message schedule meets its deadline. Most existing work, however, only considers the scheduling for either static segments [2], [8], [9], and [10] or dynamic segments [11], [12], and [13]. This isolated scheduling severely limits the performance in terms of bandwidth utilization and transmission latency.

Observation 2: The Inefficiency of Fault Tolerance

In FlexRay networks, faults may be frequent and ubiquitous due to radiation, interference and temperature variation. Such faults can be classified into permanent and transient faults [14]. Permanent faults are usually caused by physical damages and lead to long-term malfunctioning. Transient faults usually result in the miscalculations in the logic and data corruption and last for a short duration. *X-by-wire* automotive applications are safety-critical. They require data integrity even with the occurrence of transient faults. Moreover, with the increasing numbers of rich electronic devices in cars (e.g., around 2500 signals are exchanged among 70 ECUs of luxury cars [5], [15]), handling transient faults demands efficient fault-tolerant techniques to improve the system reliability. Fault tolerance in an important feature of FlexRay. Existing work offers fault tolerance mainly using scheduling and probabilistic analysis. Although authors in [16] formulated the scheduling problem as a mixed integer linear programming algorithm, its design goal was to re-transmit as many faulty messages as possible, which can not offer reliability guarantee due to choosing the re-transmitted messages in an ad-hoc manner. The re-transmission of faulty messages can ameliorate the reliability with extra load in the bandwidth and additional transmission latency. Recently, authors in [9] used a systematic probabilistic analysis to provide formal guarantee on desired reliability levels. However, this work only considers the static segments of FlexRay.

Observation 3: The Gap Between Real-Time Communications in Networks and Storage Management in End Devices

In general, FlexRay-based work mainly studies the approaches to optimize communication performance. In practice, the performance also tightly depends upon the storage management in end devices. Extending the end-to-end transmission to the end devices is important to deliver high performance and improve system reliability. To address this problem and bridge the gap between real-time communication and storage systems, we need to first select a storage device that can meet the needs of the transmission characteristics of FlexRay. Flash-based solid state drive (SSD) is a good candidate. SSD is rapidly being integrated into modern computer systems. It has the salient properties of random addressing, high performance (read/write), energy efficiency, small size and shock resistance. While SSD offers space and cost advantages over DRAM and performance and energy advantages over hard disks, it in practice suffers from limited utilization [17], [18], [19], and [20]. To alleviate the limited utilization of SSD (in terms of lifespan and performance), we argue that deduplication is an efficient solution that is actually overlooked in FlexRay-based transmission systems.

Existing work fails to efficiently address the above challenges. Specifically, although FlexRay has been widely used as an in-vehicle communication network, its applicability is severely hindered in high-speed safety-critical X-by-wire systems [21]. FlexRay does not provide acknowledgement or re-transmission schemes and hence there is limited guarantee on message delivery for reliability. Moreover, the authors in [16] formulated the scheduling problem as a mixed integer linear programming algorithm, but its design goal was to re-transmit as many faulty messages as possible, which may fail to offer reliability guarantee due to that the re-transmitted messages are chosen in an ad-hoc manner. The re-transmission of faulty messages can improve the reliability with extra loads in the bandwidth and additional transmission latency. Recently, authors in [9] uses systematic probabilistic analysis to provide formal guarantee on desired reliability levels. However, this work only considers the static segments of FlexRay.

To address the above challenges, we propose a novel holistic scheduling scheme, called HOSA. HOSA considers both SS and DS in the holistic scheduling design and can support scalable fault tolerance, improve bandwidth utilization and support efficient storage in end devices. Specifically, we make the following contributions.

1.1 Holistic Scheduling

FlexRay is a high-bandwidth communication protocol with a cyclic operation. Each FlexRay cycle consists of a static segment and a dynamic segment. The former is designed for the periodic transmission of real-time data, while the latter supports the transmission of low-priority data and event-triggered (aperiodic) real-time data. HOSA employs a novel holistic scheme to schedule both static segments and dynamic segments in a unified manner. Fast and accurate slot computation allows HOSA to identify available static slots that can be pilfered by the task that transmits dynamic messages. Idle slots are hence minimized and HOSA achieves high bandwidth utilization.

1.2 Scalable Fault Tolerance

Scalable fault tolerance refers to the ability of the FlexRay protocol to operate in the configurations that provide various degrees of fault tolerance. HOSA is compliant with existing schemes for scalable fault tolerance, and focuses on flexibly scheduling dual channel communication and efficiently optimizing bandwidth utilization. HOSA implements this through the design of a novel slot pilfering technique. It further leverages approximation computation to significantly reduce the complexity with slight impact on the available pilfered slots.

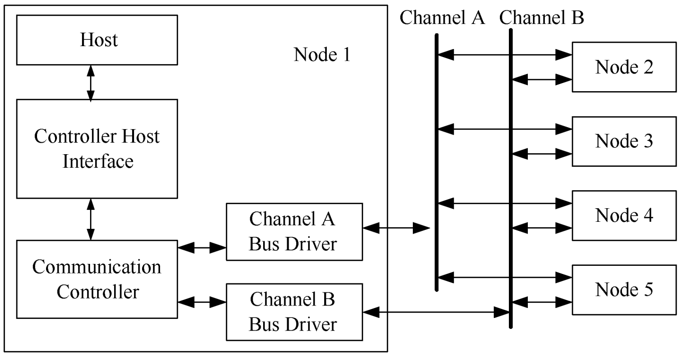


Fig. 1. Illustration of a FlexRay cluster.

1.3 Real System Implementation

To examine the performance of our proposed HOSA scheme in FlexRay networks, we implement HOSA, including SSD prototype and its deduplication-aware FTL, in a real testbed. The prototype contains all the mentioned components and functionalities. Since many of the internal features, say, physical-page allocation and data granularity, are hidden, existing work often considers SSD as a black box. Based on our previous implementations in SSD, including HAT [22] and SSDsim [23], we further implement and improve the SSD prototype with the content-based FTL to make it suitable to inline deduplication. In the meantime, the SSD parallelism is further explored and exploited to support the operations. Moreover, we use synthetic test cases and real-world case studies from the automotive industry to evaluate the system performance in terms of overall running time, bandwidth utilization, deadline miss ratio, transmission latency and deduplication. Experimental results demonstrate the efficiency and efficacy of HOSA. For instance, compared with the existing industrial FlexRay implementation [3], HOSA obtains about 50 percent improvements on bandwidth utilization and 61.5 percent reduction in transmission latency.

The rest of this paper is organized as follows. Section 2 presents the FlexRay scheduling model. Section 3 describes the holistic scheduling on both static and dynamic segments. Section 4 presents the SSD based implementation to offer content-based management for inline deduplication. We present the performance evaluation in Section 5. Finally, we conclude our paper in Section 6.

Compared with our conference version [24], we make significant improvements by extending the end-to-end analysis to the end devices. We leverage deduplication-aware SSD as a suitable storage device to meet the needs of real-time FlexRay processing requirements. The extended work also bridges the gap between real-time transmission and storage management in FlexRay.

2 FlexRay Scheduling Model

2.1 FlexRay Cluster

A FlexRay cluster consists of the network nodes connected by FlexRay communication channels as shown in Fig. 1. FlexRay allows a cluster to be flexible configuration of network topology, such as bus, star or hybrid connection. A cluster is a communication system that contains multiple nodes connected via at least one communication channel directly in a bus topology or by star couplers in a star topology. Moreover, each node in a FlexRay cluster consists of a host and a communication controller (CC), which are connected by a controller-host interface (CHI). The host is a part of an Electronic Control Units (ECU) where the application software is executed to handle incoming messages and generates outgoing messages. The communication controller implements the FlexRay protocol services. CHI serves as a buffer between the host and the CC.

To support real-time message communication, a bus driver that has a transmitter and a receiver connects with the communication controller to one communication channel that supports the inter-node connection. The bus driver also maintains clock synchronization with other nodes, constructs and checks cyclic redundancy code verification. The network nodes thus exchange periodic and aperiodic real-time messages that are transmitted in FlexRay communication cycles.

2.2 Static and Dynamic Segments

Static and dynamic segments are the structures of message delivery in a FlexRay network. Static segment is a portion of the communication cycle where the media access is controlled via a TDMA scheme. FlexRay can determine the access to the media in a static segment only by the progression of time. Furthermore, the dynamic segment portion of the communication cycle makes use of Flexible Time Division Multiple Access (FTDMA) to schedule the media access via a minislotting scheme. The minislot is a time interval of the dynamic segment to support flexible timing configuration. FlexRay then allows the dynamic segment access to the media based on a priority manner for the nodes to transmit data.

Static and dynamic segments demonstrate different formats and functionalities in the communication slots. Specifically, static communication slot is an interval of time. The access to a communication channel is allowed exclusively to a specific node for transmitting a frame with a frame ID that corresponds to the slot. Each static communication slot contains a constant number of *macroticks* regardless of whether or not a frame is sent in the slot. In the static segment, all communication slots are of identical and static configuration.

Furthermore, dynamic communication slot contains one or more *minislots*. The smallest time unit in a DS is the minislot with a duration representation of $gdMinislot$. A DS contains a maximum number of $gNumberOfMinislots$ (between 0 and 7986) minislots. Unlike a static communication slot, FlexRay allows the duration of a dynamic communication slot to vary depending on the length of the frame. A variable $vSlotCounter$ contains the ID of the current dynamic slot starting from a pre-configured value. In each dynamic slot, a frame with the corresponding ID is transmitted, and hence the duration of the dynamic slot is determined by the length of the transmitted frame. If no frame is sent, the duration of a dynamic communication slot is equal to that of one minislot. In fact, frames are transmitted within dynamic slots that are superimposed on the minislots.

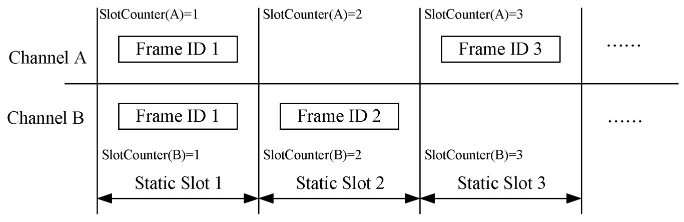


Fig. 2. Scheduling static segments.

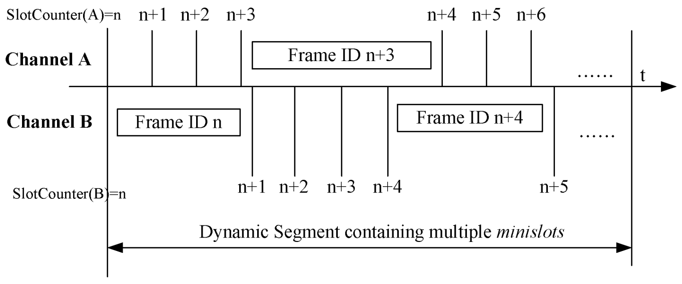


Fig. 3. Scheduling dynamic segments.

2.3 Dual Channel Design

Dual channel design in the FlexRay specification [3] offers flexible transmission patterns for the static and dynamic segments. Specifically, for scheduling static segments, each network node maintains a slot counter variable $SlotCounter(A)$ for channel A and a slot counter variable $SlotCounter(B)$ for channel B. Both slot counters are initialized with 1 at the beginning of each communication cycle and further incremented at the end of each communication slot.

Fig. 2 illustrates the transmission patterns in a single node that makes use of the static segments. The scheduling on static segments depends upon the operations defined in a schedule table. For example, in slot 1 the node transmits a frame on channel A and a frame on channel B. In slot 2 the node transmits a frame only on channel B. For scheduling dynamic segments, each network node maintains two slot counters, respectively for channels A and B, in scheduling the dynamic segments. Fig. 3 illustrates the scheme of scheduling the dynamic segments. Note that although the slot counters for channel A and for channel B are incremented simultaneously within the static segment, their values can be incremented independently according to the dynamic arbitration scheme.

Fig. 4 shows the periodic communication that has two cycles of length in channel A and B. Each cycle contains two time intervals with different access policies (a static and a dynamic segment). They have different lengths that are fixed over the cycles. Moreover, both the static and dynamic segments have multiple slots. In the static segment, FlexRay allows the slots number to be fixed. The length of these slots are constant and equal, regardless of whether static messages are sent or not in that cycle. FlexRay uses the global configuration parameter $gdStaticSlot$ to specify the length of a static slot [3]. As shown in Fig. 4, there are four static slots for the static segment. Note that a FlexRay cycle generally contains a symbol window and a network idle time. Since they are actually not related with our scheduling analysis, for simplicity, we ignore them in the examples.

The performance in practical FlexRay networks relies on the definition of the dynamic segments' lengths. FlexRay specifies the length of the dynamic segment in the number of "minislots", which is equal to $gNumberOfMinislots$. During the transmission of dynamic segments, if there is no message to be sent during a slot, the length of this slot becomes very small. Otherwise, the dynamic slot offers a transmission length, i.e., the number of minislots, to allow for transmitting the whole message.

At the beginning of each communication cycle, the communication controller of a node resets the counters of slots and minislots for initialization configuration. Moreover, the controller also needs to check if there exist messages to be transmitted, which will be further organized into the frames. As shown in Fig. 4, there exists an assumption that all messages to be transmitted are ready before the first bus cycle. In practice, due to different schemes in scheduling static and dynamic segments, the transmission scenarios would be different. Specifically, static segments use a schedule table to select the messages into static frames to transmit in the bus cycle. For example, messages M_l and M_p are placed into the associated static buffers in the CHI to be transmitted in the first bus cycle.

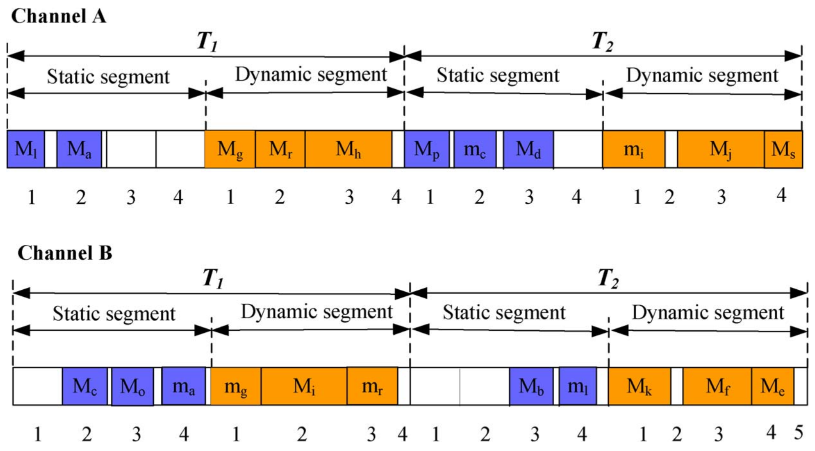


Fig. 4. Dual channel scheduling on static and dynamic segments.

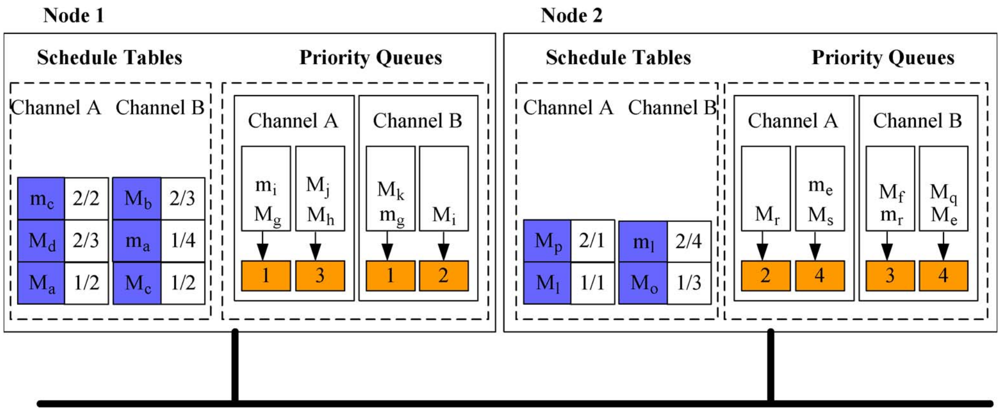


Fig. 5. Node architecture for static and dynamic segments in a dual channel.

Moreover, transmitting a dynamic message is constrained and conditional. Only if there exist enough idle slots until the end of the dynamic segment, the selected messages can be transmitted during the dynamic segment of the bus cycle. In the real implementations, when the dynamic slot counter reaches the value of the Frame ID of the transmitted message, FlexRay needs to check if the current value of the minislot counter is smaller than a given value $pLatestTx$. For each network node, the value $pLatestTx$ is fixed and depends upon the size of the largest dynamic frame. For example, message M_e prepares for transmission before the first bus cycle starts. However, after message m_r is transmitted, there are not enough slots left in the dynamic segment. This will delay the transmission of M_e for the next bus cycle.

3 Holistic Scheduling

This Section presents the holistic scheduling for both static and dynamic segments with the aid of slot pilfering technique.

3.1 FlexRay Node Architecture

A FlexRay network supports dual-channel communication to offer the guarantee of transmission reliability. Fig. 5 shows the node architecture for scheduling static and dynamic segments in the dual channel. Each node uses a *schedule table* to maintain and schedule the messages to be transmitted in the static segments, while using *priority queues* for the dynamic segments. Here, we use capital letter to represent the original messages and lower case for corresponding redundant ones.

Static and dynamic segments have different scheduling schemes to allow a message to be transmitted. First, for scheduling static segments, a message in the schedule table has a timing based sequence, i.e., the number of cycles and slots. For example, the message M_a is transmitted on the second slot of the first cycle, represented as "1/2". On the other hand, for dynamic segments, we allocate the slot number to each node and all messages in each priority queue will be scheduled in the fixed priority way. For example, node 1 sends the messages to slots 1 and 3 of channel A. For each of these slots, CHI provides a buffer that can be written by the host and read by the communication controller. At the beginning of each slot, the communication controller needs to read the messages in the buffers so as to facilitate the transmission of frames.

To significantly reduce the potential transmission collision and obtain the performance improvement, during any communication slot, FlexRay only allows one node to send messages on the bus. This node needs to transmit the message with the frame ID that is equal to the current value of the slot counter. We set two slot counters that respectively correspond to the static and dynamic segments. In the design phase, we decide and allocate the frame identifiers to nodes. Each node to send messages has one or more static and/or dynamic slots.

For static and dynamic messages, we further leverage different schemes to decide which messages are transmitted during the allocated slots. For static messages, there exists a schedule table with the transmission time in each network node. When transmitting a static message starts, a given message is placed into its associated static buffer in the CHI. For example, static message M_a sent from node 1 has an entry "1/2" in the schedule table specifying that it should be sent in the second slot of the first static cycle.

On the other hand, for scheduling dynamic messages, there is an assumption that Frame ID is specified in advance. For example, as shown in Fig. 5, dynamic message M_h has the frame identifier "3". Moreover, FlexRay allows a node to send different messages using the same dynamic Frame ID. For example, messages M_j and M_h on node 1 have both Frame ID 3. If two or more messages with the same frame ID prepare to be sent in the same bus cycle, a priority scheme is used to decide which message will be sent first. By considering the dual-channel transmission, each dynamic message M_i or m_i has their associated priority, say $priority_{M_i}$ or $priority_{m_i}$. Messages with the same Frame ID will be inserted into a local output queue, in which we order them based on their priorities. The message from the head of the priority queue will be sent in the current bus cycle. For example, message M_h will be sent before M_j because it has a higher priority.

In addition, original and redundant messages may be not identical in the receiver node, although this case occurs with very small probability. In this case, the receiver node will require a retransmission.

3.2 Holistic Scheduling Segments

To optimize the bandwidth utilization and offer substantial performance improvements, we use holistic scheduling upon the static and dynamic segments in the FlexRay network. Specifically, we consider the transmission of static and dynamic segments respectively as hard deadline periodic and soft deadline aperiodic tasks. The design goal is to *schedule a mixture of periodic and aperiodic tasks in a dual channel to guarantee that all periodic deadlines are met and the response time for the aperiodic tasks can be as small as possible in the FlexRay network*. Holistic scheduling scheme hence offers available time for completing the aperiodic tasks by "*pilfering*" all the time from the periodic tasks without causing their deadlines to be missed.

The main idea behind slot pilfering comes from the practical observations and long-term experiences. When an aperiodic request arrives, the slot judiciously pilfers all the available slots from periodic tasks, which are used to satisfy the aperiodic requests. On the other hand, when there are no pending aperiodic requests, we schedule the periodic tasks as usual. We further formulate the slot pilfering technique in a FlexRay network that contains n periodic tasks, $\tau_1, \tau_2, \ldots, \tau_n$.

Definition 1. *Each task, τ_i $(1 \leq i \leq n)$, is denoted by a 4-tuple $\tau_i = \{C_i, T_i, \phi_i, d_i\}$, where C_i is the worst-case computation requirement, T_i is a period, ϕ_i $(0 \leq \phi_i \leq T_i)$ is an offset relative to time origin, and d_i $(d_i \leq T_i)$ is a hard deadline. We assume that the parameters C_i, T_i, ϕ_i and d_i, are the known deterministic quantities.*

A fixed priority algorithm, say deadline monotonic algorithm [25], can schedule these tasks. In the meanwhile, the tasks with smaller value of d_i are allocated higher priority.

We leverage differentiated representation for scheduling the tasks of static and dynamic segments. For a periodic task τ_i for the static segment, it leads to an infinite sequence of jobs. We further consider the scheduling on aperiodic tasks for dynamic segments as the problem of parameter optimization. Specifically, we place an aperiodic task for a dynamic segment into the queues based on deadline orders. The slot value, associated with an enqueued aperiodic task for the dynamic segment, demonstrates how many available slots can be allocated to facilitate its processing, while its deadline is still met. To achieve this goal, we need to use the value of available slots to offer transmission guarantee. New aperiodic tasks for dynamic segments will not incur its deadline to be violated. At the same time, all periodic deadlines for scheduling static segments are also guaranteed. Therefore, the remaining processing time can be competed between hard and soft aperiodics. We further describe the aperiodic task for scheduling a dynamic segment.

Definition 2. *The aperiodic task J_k for scheduling a dynamic segment is represented as a 3-tuple, $J_k = \{\alpha_k, p_k, D_k\}$, where α_k is the associated arrival time, p_k is the processing requirement and D_k is the hard deadline. To support the retrieval of aperiodic tasks, HOSA defines that $0 \leq \alpha_k \leq \alpha_{k+1},\ k \geq 1$.*

Based on the above definitions, HOSA aims to minimize the response time of J_k, represented as R_k. Specifically, we consider $W(t) = \sum_{k|\alpha_k \leq t} p_k$ as a cumulative aperiodic workload process. This process collects all the aperiodic tasks. These tasks share the same property of the arrival time within the interval $[0, t]$. Moreover, a cumulative aperiodic execution process, ε_t, is a continuous function with the property of $\varepsilon_t \leq W(t),\ t \geq 0$. HOSA thus describes the completion time of J_k, as $T_k = \min\{t|\varepsilon_t = \sum_{i=1}^{k} p_i\}$. Therefore, HOSA achieves $R_k = T_k - \alpha_k$.

To efficiently support the holistic scheduling on the static and dynamic segments in the dual channel, we need to determine the maximum processing time that can be pilfered from hard deadline periodic tasks. FlexRay communication system can use a slot pilferer to schedule the static and dynamic segments. The slot pilferer can efficiently address the problem of minimizing the response times of soft aperiodic tasks, while offering the guarantee that the deadlines of hard periodics are also met.

For real-time networked systems, packets are usually modeled as tasks. There are preemptive tasks and non-preemptive tasks. Un-interruptible packets can be modeled as non-preemptive tasks. In the context of our proposed work, the performance metric we examine is the network transmission time. We hence use the preemptive task model. Even for the case when network transmission can not be interrupted, we can recourse to non-preemptive task models, through techniques such as adding blocking times in the analysis [26].

4 Cost-Effective Deduplication in SSD-Based End Devices

Deduplication functionality in FlexRay generally needs to handle massive streaming data in a real-time manner, while obtaining space savings. In practice, the index structure of massive data easily overflows the DRAM size and has to be stored on hard disk. The operations upon index structure are thus limited by expensive and slow seek and switch operations upon mechanical-based disks. In the flash, different from disks, random read operations are comparable to sequential read operations due to no mechanical head movements. Since deduplication needs to be completed timely, it is desirable to obtain high throughput in inline deduplication systems and meet the needs of fast detecting duplicate segments. To address this problem, SSD is a good candidate due to its salient features with respect to system performance, such as energy efficiency, read/write latency and shock resistance.

To provide efficient data management, we need to carefully design the data allocation scheme. The function of allocation scheme is to decide the mapping between physical page and logical page. When a new write request arrives, the allocation scheme chooses a free physical page based on the factors of idle/busy states of channels and chips, the erased count of blocks and the priority order of parallelism.

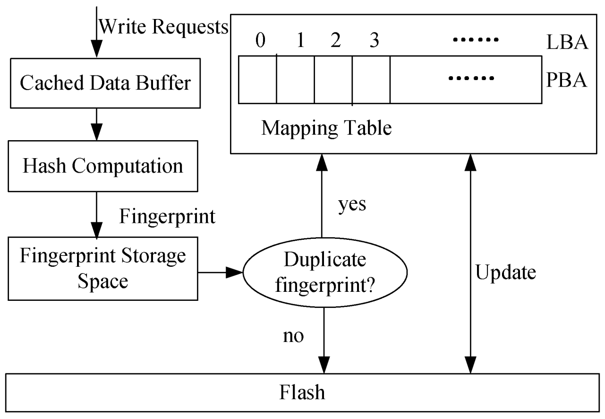


Fig. 6. Deduplication workflow in HOSA.

4.1 Design Principles

HOSA can work for inline deduplication. Inline deduplication refers that HOSA proactively examines the incoming data and removes duplicate writes before committing a write request to flash. To support inline deduplication in a FlexRay network, HOSA is designed for running in an SSD device that has limited memory space and computing power. This matches the properties of FlexRay end devices. Moreover, different from backup streams with high redundancy, HOSA generally handles regular system workloads that exhibit relatively lower duplication rate. HOSA further offers high real-time data access performance, rather than running during out-of-office hours in backup systems.

To reduce write traffic to flash memory and obtain space savings in SSD, HOSA eliminates unnecessary duplicate writes, which further improves the efficiency of garbage collection and wear-leveling. HOSA leverages chunk-based deduplication approach, in which a fixed-sized chunking is used. The reasons come from the observations that in a FlexRay network, the sizes of sequential requests are generally very small and the basic operation unit in flash is a page (say 4 KB). The internal management policies in SSDs, such as the mapping policy, are also designed in units of pages. Hence, to avoid unnecessary complexity, we leverage pages as the fixed-sized chunks for hashing computation.

To accelerate the identification of duplicate writes, existing approaches mainly rely on the exploration of temporal and spatial localities [22], [27], [28], [29], [30], [31], and [32]. For example, temporal locality helps buffer writes to reduce duplicate writes to SSD. Spatial locality helps aggregate multiple subpage writes into fewer page writes. Unlike them, HOSA leverages content locality that preferentially accesses some contents and facilitates data deduplication. By exploiting and exploring the contents of data, HOSA offers the content-aware FTL that mainly consists of fingerprint generation, fingerprint lookup and mapping management. The fingerprints of FlexRay segments are the hash values that can summarize the content of written segments. The generated fingerprints are then used for the lookups of duplicate segments. The mapping management handles the mapping between the host-viewable logical addresses and physical flash addresses. Note that the content-aware FTL in the HOSA design is an augmentation, rather than a complete replacement, to the existing FTL designs. The content-aware FTL is orthogonal to the other FTL policies, such as the garbage collection and wear leveling policies.

4.2 Prototype Implementation

To implement the deduplication functionality, HOSA intercepts incoming write requests at the SSD device level and uses a hash function to generate fingerprints that can summarize the content of updated data. By querying a fingerprint storage space, which maintains the fingerprints of stored data in SSD, HOSA can accurately and efficiently eliminate duplicate writes to flash.

In general, SSD exposes an array of logical block addresses (LBA) to the host and uses a mapping table to show the mapping relationship between LBA and the physical block address (PBA). Fig. 6 illustrates the deduplication-aware process of handling a write request in HOSA. Specifically, if a write request arrives:

- The incoming data is first temporarily maintained in the on-device buffer;
- HOSA computes the hash value of each updated page in the buffer, called fingerprint;
- Each fingerprint is looked up against a fingerprint storage space, in which HOSA maintains the fingerprints of data already stored in the flash memory;
- If a duplicate fingerprint is found, meaning that a residing data unit contains the same content, the mapping table that carries out the translation between LBA and PBA is updated by mapping to the physical location of the residing data. HOSA in the meantime eliminates the write to flash;
- If a duplicate fingerprint is not found, HOSA executes the write to the flash memory as a regular write.

HOSA attempts to identify and remove duplicate writes. A byte-by-byte comparison is obviously slow and can not meet the needs of real-time inline deduplication in FlexRay. HOSA uses hash functions to compute a hash value as a fingerprint. Duplicate data can be determined by comparing fingerprints that summarize the contents of pages. We use the SHA-1 hash function [33], [34] due to its practically collision-resistant properties to index and compare pages. For each page, we calculate a 160-bit hash value as its fingerprint and store it as the page's metadata in flash.

To comprehensively examine the performance of the content-aware FTL, we implement it in a real SSD prototype, which is event-driven, modularly structured, and multi-tiered. It serves as an SSD hardware platform and executes the proposed FTL schemes, allocation schemes, buffer management algorithms and request scheduling algorithms. The SSD prototype consists of three tiers, including the buffer and request-scheduling module at the top, the FTL and allocation module in the middle, and the low-level hardware platform module at the bottom.

5 Performance Evaluation

In this Section, we show the experimental results of implementing our proposed HOSA running on mixed datasets (including static and dynamic segments).

TABLE 1
Brake-by-Wire Message Parameters

Message	Offset (ms)	Period (ms)	Deadline (ms)	Size (bits)
1	0.26	8	8	1280
2	0.72	8	8	272
3	0.52	1	1	1560
4	0.88	1	1	563
5	0.92	1	1	345
6	0.96	1	1	425
7	0.22	1	1	1172
8	0.27	8	8	852
9	0.76	8	8	763
10	0.39	8	8	915
11	0.91	8	8	1245
12	0.52	8	8	628
13	0.69	8	8	427
14	0.81	8	8	338
15	0.93	8	8	847
16	0.42	8	8	1560
17	0.61	1	1	1730
18	0.53	1	1	532
19	0.95	1	1	1154
20	0.77	1	1	861

TABLE 2
Configuration Parameters for Dynamic Segments

Configuration Parameter	**Value**
gdMacrotick [μs]	1
gNumberOfStaticSlots[macrotick]	80, 120
gdCycle [μs]	5000
gdStaticSlot[macrotick]	40
gdMacroPerCycle	5000
gdMinislot[macrotick]	8
gdSymbolWindow[macrotick]	0
gdDynamicSlotIdlePhase[minislot]	1
gdMinislotActionPointOffset[macrotick]	2

5.1 Experimental Configurations

Our experiments are performed using 10 FlexRay nodes that are connected to a bus analysis tool that helps record the information of message transmission in the FlexRay network. The FlexRay nodes are implemented and configured by multiple networked boards that consist of a 16-bit Flash-based controller unit to support the FlexRay protocol operations, 2 IP-modules for the dual-channel design, and FlexRay-enabled transceivers to support the physical layer of the FlexRay bus. SSD prototypes are connected with the FlexRay end nodes and we examine their real performance. To comprehensively test the performance, we initially set cache size in SSD to be 64 KB, which is further dynamically adjusted. Moreover, we use an independent module to receive and maintain all messages that are transmitted on the FlexRay bus, to facilitate real-time transmission analysis.

The experiments make use of the mixed datasets that contain both static and dynamic segments. Specifically, for the datasets of static segments, the datasets consist of synthetic test cases and one real-world scenario. The synthetic test cases were generated by varying message parameters, such as periods and deadlines, to cover a wide range of possible scenarios. The periods are varied between 2 ms and 50 ms. The deadlines are varied between 1 ms to 20 ms. The FlexRay communication cycle period is 5 ms and the static cycle length is 3 ms, based on the experiences from the industry [10]. The test cases contain a large number of messages. Moreover, we consider a real-world X-by-wire application, i.e., brake-by-wire, which has been widely used in performance evaluation of the FlexRay-based design. Table 1 shows the details of the associated parameters.

For the datasets of dynamic segments, we configure the parameters in each communication cycle. We set the values of the parameters as shown in Table 2. The suitable timing properties of aperiodic messages used in our experiments are taken from a message set that is published by the Society for Automotive Engineers [35]. Hence, we consider aperiodic messages with a period (minimum inter-arrival time) and a deadline of 50 ms. We use 30 aperiodic messages with the IDs, from 81 to 110 or from 121 to 150, respectively corresponding to the sequential numbers in 80 and 120 slots. The maximum number of their transmission slots $cSlotIDMax$ are 110 and 150, respectively.

We uniformly distribute the aperiodic messages in 10 FlexRay nodes. In each network node, an interrupt-based routine running as the host process generates the aperiodic messages. We use a 16-bit reload timer to count down the time until the next generation of each message. Furthermore, for generating the event-based messages, a $rand()$ function in C standard library computes the next generation time. Furthermore, in real-time transmission performance, we compare the HOSA scheme with the standard implementation of FlexRay specification (FSPEC) [3] and HOSA without the approximation for reducing computation complexity, in terms of overall running time, bandwidth utilization, average transmission latency for static and dynamic segments, and deadline miss ratio.

The minimum length of the dynamic segment is determined by $gNumberOfMinislots$. We select the dynamic segments with 50 and 100 minislots in our evaluation. To adjust the length of the dynamic segment, We vary the value of parameter $gNumberOfMiniSlots$. To compensate the modification of the dynamic segment length, we change the parameter $gdNIT$ (duration of network idle time) so as to keep the frame cycle duration of 5000 μs as a constant.

For evaluating deduplication performance, we compare HOSA with state-of-the-art deduplication schemes, ChunkStash [29] and SiLo [36]. SiLo is our previous work and we can examine its performance in the real SSD prototype. Since there are no open source codes of ChunkStash, we choose to re-implement it. Specifically, we have implemented the locality-based and exact-deduplication of ChunkStash, while carefully considering its design principles and algorithms [29], including the inherent locality of backup streams and cuckoo hash based indexing structure.

For fair comparisons, all schemes for comparisons use the same running environments. Furthermore, the main deduplication metrics include duplicate elimination, latency and throughput. The duplicate elimination is defined as the percentage of duplicate data eliminated. The latency records the overall consumption time of completing the deduplication operations. The throughput is the rate at which the streams are processed. It is worth noting that our evaluation testbed is not a production-quality deduplication system, but a research prototype. The results hence

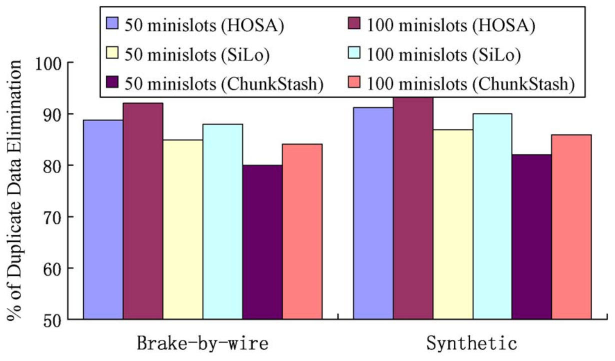


Fig. 7. Percentage of duplicate data elimination.

should be interpreted as approximate and comparative evaluation.

5.2 Deduplication Performance

5.2.1 The Percentage of Duplicate Elimination

Fig. 7 shows the percentages of duplicate data elimination. We observe that HOSA can remove on average 88.8 percent and 90.2 percent duplicate data respectively in brake-by-wire and synthetic workloads, which are larger than those of SiLo and ChunkStash. SiLo needs to exploit both similarity and locality of data segments and ChunkStash leverages cuckoo hashing that potentially leads to endless loop. Their high computation complexity results in long processing latency and the missing of continuous data segments that often contain duplicates. Unlike them, by leveraging content-aware FTL design, HOSA is able to accurately identify duplicate data. In addition, there exists slight difference between 50 and 100 minislots. The reason is that the transmission in 50 minislots incurs relatively higher deadline miss rate and more similar data are lost. The deduplication percentage hence decreases.

5.2.2 Deduplication Latency

Fig. 8 shows the deduplication latency in SSD with various cache sizes. The latency decreases when increasing the cache sizes. We observe that the decrements of deduplication latency are limited when the cache size is larger than 128 KB. Even if the sizes continue to increase, the latency has slight reduction. We argue that a suitable cache size is important to obtain a good tradeoff between space overhead and performance improvements. Moreover, the synthetic dataset, that has larger numbers and sizes of data segments than the brake-by-wire, incurs on average 10.5 ms that is larger than 4.2 ms in the latter.

HOSA obtains over 45.3 percent and 55.7 percent smaller latency respectively than SiLo and ChunkStash in 50 minislots, thus meeting the needs of real-time processing. In contrast, ChunkStash needs to maintain at least 6 bytes for each new chunk, resulting in a very large cuckoo hash table for fingerprints and consuming large fraction of limited-size cache. SiLo needs to maintain a large index structure to support the exploitation of similarity and locality. More data accesses have to visit the low-speed disks. We can obtain similar conclusion when executing 100 minislots.

5.2.3 Deduplication Throughput

Fig. 9 shows the deduplication throughput when executing the data deduplication in SSD. Specifically, in the brake-by-wire dataset, the average throughputs are 327.8 MB/s, 262.4 MB/s and 225.7 MB/s, respectively for HOSA, SiLo and ChunkStash. Compared with SiLo and ChunkStash, the advantages of HOSA come form its content-aware FTL design and space-efficient index structure for segment lookups. The similar observations can also obtained from the synthetic dataset. HOSA hence can offer high throughout for data deduplication and significantly improve the utilization of SSD.

6 Conclusion

Providing scalable fault-tolerance is important to the FlexRay networks. This paper proposes a cost-effective scheme, called HOSA, that supports holistic scheduling on both static and dynamic segments in the dual-channel communication system. HOSA leverages a slot pilfering technique to significantly minimize idle slots, improve bandwidth utilization and decrease transmission latency. HOSA efficiently handles the complexity of slot computation with the aid of a proper approximation approach. Moreover, to bridge the gap between real-time transmission in networks and storage management in end devices,

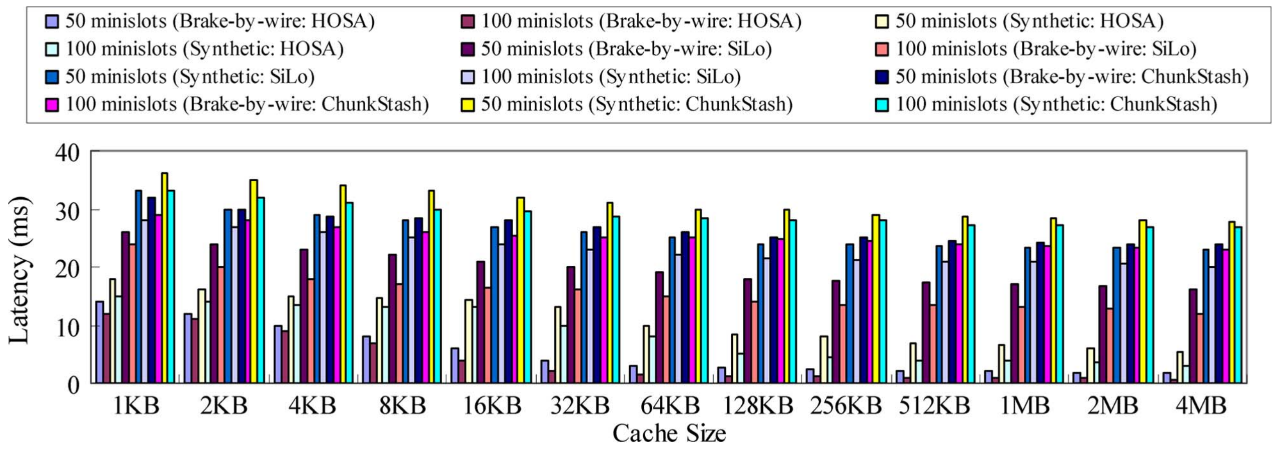


Fig. 8. Deduplication latency.

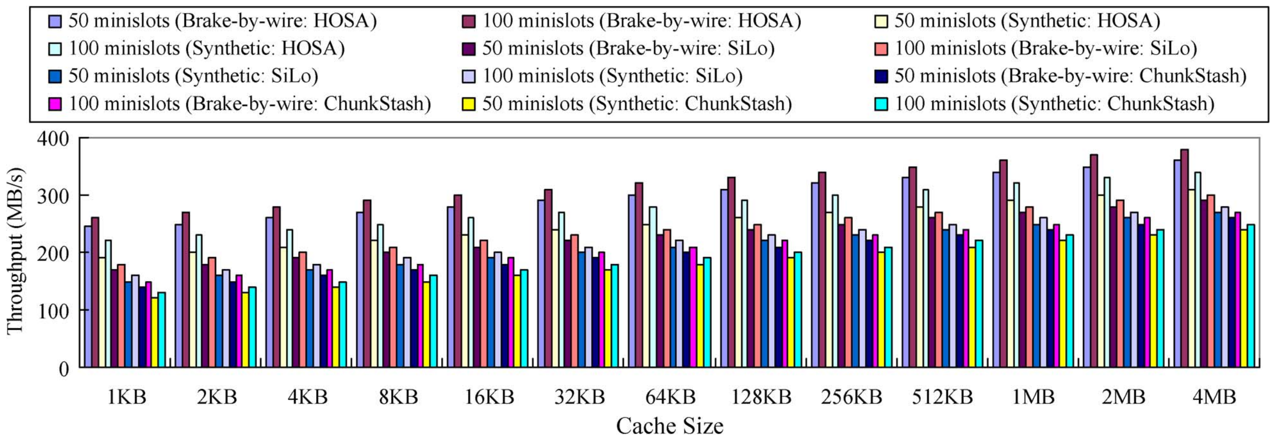


Fig. 9. Deduplication throughput.

we implement a deduplication-aware SSD design in a real prototype. Extensive experimental results based on synthetic and real-world test cases demonstrate the efficiency and efficacy of the proposed HOSA scheme.

Acknowledgment

This work was supported in part by National Basic Research 973 Program of China under Grant 2011CB302301; National Natural Science Foundation of China (NSFC) under Grant Nos. 61173043, 61025008, 61232004; the NSERC Discovery Grant 341823; US National Science Foundation Award 1116606; NSERC Grant RGPIN 418521-12; Fundamental Research Funds for the central universities, HUST, under Grant 2012QN098.

References

[1] I. Park and M. Sunwoo, "Flexray Network Parameter Optimization Method for Automotive Applications," *IEEE Trans. Ind. Electron.*, vol. 58, no. 4, pp. 1449-1459, Apr. 2011.

[2] H. Zeng, M. Di Natale, A. Ghosal, and A. Sangiovanni-Vincentelli, "Schedule Optimization of Time-Triggered Systems Communicating Over the Flexray Static Segment," *IEEE Trans. Ind. Informat.*, vol. 7, no. 1, pp. 1-17, Feb. 2011.

[3] The Flexray Communication System Specification, Version 2.1. [Online]. Available: http://www.flexray.com

[4] B. Brake System Relies on FlexRay. [Online]. Available: http://www.automotivedesignline.com/news/218501196, July 2009.

[5] N. Navet, Y. Song, F. Simonot-Lion, and C. Wilwert, "Trends in Automotive Communication Systems," *Proc. IEEE*, vol. 93, no. 6, pp. 1204-1223, June 2005.

[6] H. Kopetz and G. Bauer, "The Time-Triggered Architecture," *Proc. IEEE*, vol. 91, no. 1, pp. 112-126, Jan. 2003.

[7] Cena, Gianluca, Valenzano, and Adriano, "Performance analysis of Byteflight networks *Proc. IEEE Int. Workshop Factory Commun. Syst.*, pp. 157-166, 2004.

[8] K. Schmidt and E. Schmidt, "Message Scheduling for the Flexray Protocol: The Static Segment," *IEEE Trans. Veh. Technol.*, vol. 58, no. 5, pp. 2170-2179, June 2009.

[9] B. Tanasa, U.D. Bordoloi, P. Eles, and Z. Peng, "Scheduling for Fault-Tolerant Communication on the Static Segment of FlexRay," in *Proc. IEEE Real-Time Syst. Symp.*, 2010, pp. 385-394.

[10] M. Lukasiewycz, M. Glaß, J. Teich, and P. Milbredt, "Flexray schedule optimization of the static segment," in *Proc. CODES+ISSS*, 2009, pp. 1-10.

[11] E. Schmidt and K. Schmidt, "Message Scheduling for the Flexray Protocol: The Dynamic Segment," *IEEE Trans. Veh. Technol.*, vol. 58, no. 5, pp. 2160-2169, June 2009.

[12] K. Schmidt, E.G. Schmidt, A. Demirci, E. Yuruklu, and U. Karakaya, "An Experimental Study of the FlexRay Dynamic Segment," in *Proc. Adv. Automot. Control*, 2010, pp. 413-418.

[13] K. Jung, M. Song, D. Lee, and S. Jin, "Priority-Based Scheduling of Dynamic Segment in FlexRay Network," in *Proc. Int'l Conf. Control, Autom. Syst.*, 2008, pp. 1036-1041.

[14] H. Kopetz, R. Obermaisser, P. Peti, and N. Suri, "From a Federated to an Integrated Architecture for Dependable Real-Time Embedded Systems," Technische Univ. Vienna, Vienna, Austria, Tech. Rep., 2004.

[15] A. Albert, "Comparison of Event-Triggered and Time-Triggered Concepts with Regard to Distributed Control Systems," in *Proc. Embedded World*, 2004, vol. 2004, pp. 235-252.

[16] W. Li, M. Di Natale, W. Zheng, P. Giusto, A. Sangiovanni-Vincentelli, and S. Seshia, "Optimizations of an Application-Level Protocol for Enhanced Dependability in Flexray," in *Proc. Design, Autom. Test Eur.*, 2009, pp. 1076-1081.

[17] Y. Wang, D. Liu, Z. Qin, and Z. Shao, "An Endurance-Enhanced Flash Translation Layer Via Reuse for Nand Flash Memory Storage Systems," in *Proc. DATE*, 2011, pp. 1-6.

[18] Y. Hua, B. Xiao, D. Feng, and B. Yu, "Bounded LSH for Similarity Search in Peer-to-Peer File Systems," in *Proc. 37th ICPP*, 2008, pp. 644-651.

[19] Z. Qin, Y. Wang, D. Liu, and Z. Shao, "Real-Time Flash Translation Layer for Nand Flash Memory Storage Systems," in *Proc. IEEE RTAS*, 2012, pp. 35-44.

[20] Z. Qin, Y. Wang, D. Liu, and Z. Shao, "A Two-Level Caching Mechanism for Demand-Based Page-Level Address Mapping in Nand Flash Memory Storage Systems," in *Proc. IEEE RTASfs*, 2011, pp. 157-166.

[21] Y. Sedaghat and S. Miremadi, "Categorizing and Fnalysis of Activated Faults in the Flexray Communication Controller Registers," in *Proc. IEEE Eur. Test Symp.*, 2009, pp. 121-126.

[22] Y. Hu, H. Jiang, D. Feng, L. Tian, S. Zhang, J. Liu, W. Tong, Y. Qin, and L. Wang, "Achieving Page-Mapping FTL Performance at Block-Mapping FTL Cost by Hiding Address Translation," in *Proc. IEEE MSST*, 2010, pp. 1-12.

[23] Y. Hu, H. Jiang, D. Feng, L. Tian, H. Luo, and S. Zhang, "Performance Impact and Interplay of SSD Parallelism Through Advanced Commands, Allocation Strategy and Data Granularity," in *Proc. ACM ICS*, 2011, pp. 96-107.

[24] Y. Hua, X. Liu, and W. He, "HOSA: Holistic Scheduling and Analysis for Scalable Fault-Tolerant Flexray Design," in *Proc. IEEE INFOCOM*, 2012, pp. 1233-1241.

[25] J. Lehoczky, L. Sha, and Y. Ding, "The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior," in *Proc. Real-Time Syst. Symp.*, 1987, pp. 166-171.

[26] X. Liu, Q. Wang, W. He, M. Caccamo, and L. Sha, "Optimal Real-Time Sampling Rate Assignment For Wireless Sensor Networks," *ACM Trans. Sensor Netw. (TOSN)*, vol. 2, no. 2, pp. 263-295, 2006.

[27] D. Andersen and S. Swanson, "Rethinking Flash in the Data Center," *IEEE Micro*, vol. 30, no. 4, pp. 52-54, 2010.

[28] G. Wu and X. He, "Delta FTL: Improving SSD Lifetime Via Exploiting Content Locality," in *Proc. EuroSys*, 2012, pp. 253-266.

[29] B. Debnath, S. Sengupta, and J. Li, "ChunkStash: Speeding Up Inline Storage Deduplication Using Flash Memory," in *Proc. USENIX ATC*, 2010, pp. 1-16.

[30] J. Kim, C. Lee, S. Lee, I. Son, J. Choi, S. Yoon, H.U. Lee, S. Kang, Y. Won, and J. Cha, "Deduplication in SSDS: Model and Quantitative Analysis," in *Proc. MSST*, 2012, pp. 1-12.

[31] A. Gupta, R. Pisolkar, B. Urgaonkar, and A. Sivasubramaniam, "Leveraging Value Locality in Optimizing Nand Flash-Based Ssds," in *Proc. FAST*, 2011, p. 7.

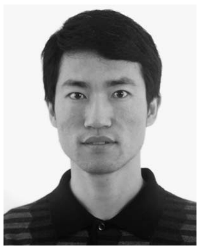
[32] F. Chen, T. Luo, and X. Zhang, "CAFTL: A Content-Aware Flash Translation Layer Enhancing the Lifespan of Flash Memory Based Solid State Drives," in *Proc. FAST*, 2011, pp. 1-14.

[33] X. Wang, Y. Yin, and H. Yu, "Finding Collisions in the Full Sha-1," in *Proc. Adv. Cryptology-CRYPTO*, 2005, pp. 17-36.

[34] C. De Canniere and C. Rechberger, "Finding Sha-1 Characteristics: General Results and Applications," in *Proc. Adv. Cryptology-ASIACRYPT*, 2006, pp. 1-20.

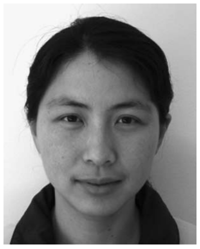
[35] "Class C Application Requirements, SAE J2056/1," in *SAE Handbook*, vol. 2. Warrendale, PA, USA: SAE, June 2013, pp. 23.366-23.371.

[36] W. Xia, H. Jiang, D. Feng, and Y. Hua, "SiLo: A Similarity-Locality Based Near-Exact Deduplication Scheme With Low Ram Overhead and High Throughput," in *Proc. USENIX ATC*, 2011, pp. 26-28.



Yu Hua received the BE and PhD degrees in computer science from the Wuhan University, China, in 2001 and 2005, respectively. He is an Associate Professor at the Huazhong University of Science and Technology (HUST), China. His research interests include computer architecture, cloud computing and network storage. He has more than 50 papers to his credit in major journals and international conferences including *IEEE Transactions on Computers (TC)*, *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, USENIX ATC, INFOCOM, SC, ICDCS, ICPP and MASCOTS. He has been on the organizing and program committees of multiple international conferences, including INFOCOM, ICPP and IWQoS. He is a Senior Member of the IEEE, and a member of ACM and USENIX.

Xue Liu received the BS degree in mathematics and M.S. degree in automatic control both from Tsinghua University, China, and the PhD in computer science from the University of Illinois at Urbana-Champaign, Champaign, IL, USA, in 2006. He is an Associate Professor in the School of Computer Science at McGill University. His research interests are in computer networks and communications, smart grid, real-time and embedded systems, cyber-physical systems, data centers, and software reliability. His work has received the Year 2008 Best Paper Award from IEEE Transactions on Industrial Informatics, and the First Place Best Paper Award of the ACM Conference on Wireless Network Security (WiSec 2011). He serves as an associate editor of the *IEEE Transactions on Parallel and Distributed Systems (TPDS)* and editor of the *IEEE Communications Surveys & Tutorials*. He is a member of the ACM and IEEE.



Wenbo He received the PhD degree from University of Illinois at Urbana-Champaign, Champaign, IL, USA, in 2008. He is currently an Assistant Professor in School of Computer Science at McGill University. She was an Assistant Professor in Department of Electrical Engineering at University of Nebraska-Lincoln from 2010 to 2011. She was with the Department of Computer Science at University of New Mexico from 2008 to 2010. Her research focuses on Pervasive Computing, and Privacy-preserving Techniques, etc. During August 2000 to January 2005, he was a software engineer in Cisco Systems, Inc. Dr. He received the Mavis Memorial Fund Scholarship Award from College of Engineering at UIUC in 2006, and the C.W. Gear Outstanding Graduate Award in 2007 from the Department of Computer Science at UIUC. She is also a recipient of the Vodafone Fellowship from 2005 to 2008, and the NSF TRUST Fellowship in 2007 and 2009. She is a member of the IEEE.

Dan Feng received the BE, ME, and PhD degrees in computer science and technology from the Huazhong University of Science and Technology (HUST), China, in 1991, 1994, and 1997, respectively. She is a Professor and Vice Dean of the School of Computer Science and Technology, HUST. Her research interests include computer architecture, massive storage systems, and parallel file systems. She has more than 100 publications to her credit in journals and international conferences, including *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, JCST, USENIX ATC, FAST, ICDCS, HPDC, SC, ICS and ICPP. She servers as the program committee member of SC 2011, 2013 and MSST 2012. She is a member of the IEEE.