Theoretical Analysis of Various Software-Defined Multiplexing Codes

Elaine Y.-N. Sun, Hsiao-Chun Wu, *Fellow, IEEE*, and Scott C.-H. Huang, *Member, IEEE*

***Abstract*— How to combine multiple data-streams for transmission in aggregate is a very interesting problem, especially for the emerging software-defined networks nowadays. The conventional packet-based protocols cannot provide the flexibility for combining data-streams in the ad hoc nature. If the number of data-streams changes over time, the existing packet formats cannot handle the transmission of multiple data-streams effectively. The effectiveness is measured by two performance metrics, namely coding efficiency and data-transmission intermittency. We propose a new software-defined multiplexing code (SDMC) approach, which can combine (multiplex) multiple data-streams easily and is much more effective than the conventional packet-based method. Three SDMC schemes (distributed, hierarchical, and hybrid) are compared theoretically and by simulation. A trade-off between these two performance metrics can be found when one selects one of the three SDMC schemes for combining multiple data-streams. The hierarchical SDMC scheme brings about the highest coding efficiency while the hybrid SDMC scheme suffers from the smallest overall intermittency.**

***Index Terms*— Software-defined multiplexing code (SDMC), software-defined network (SDN), coding efficiency, data-transmission intermittency.**

I. Introduction

CONVENTIONALLY, a network relies on communication devices subject to fixed configuration. Very often, the corresponding latency is considerable whenever the data plane requires frequent updates. Recently, *software-defined networks* (SDNs) emerge and they can separate the control plane from the data plane so that the network *programmability* or *reconfigurability* is enhanced greatly. Henceforth, the new SDN strategy enables local users to dynamically control some network functions for convenience sake. This advantage makes SDN much more flexible in managing the network efficiently in real time. Because SDNs are much more convenient to local users than the conventional networks, exciting opportunities emerge for new ideas and concepts as SDNs become popular. In this work, we would like to address the issue about "dynamic information fusion" or "data-stream multiplexing", which is constantly encountered by any network in practice. Nowadays, the data flow in a network is getting larger and larger. Moreover, high throughput, high security, and low latency are essential for all networks. Hence the optimization of traffic and efficiency becomes the popular research interest pertinent to SDNs. The effective (with low complexity) and robust (with excellent scalability) schemes for information fusion or data-stream multiplexing can well tackle these two core SDN problems.

Between the control plane and the data plane, "OpenFlow" is a common network protocol, which is simply based on *forwarding table management*. An OpenFlow switch contains the flow tables which are used to determine the path of network packets. To avoid the data overflow and the heavy traffic from time to time, the reduction of the latency induced by table matching operations is always in demand. An analytical model is established to optimize the network performance by adjusting *data-rate* and *table-occupancy* [1]. Besides, variable flow-tables in size or format are proposed for cost-effective and computationally-efficient network reconfiguration [2], [3]. The reduction of the flow entries which are in the flow table and determined by the network controller has been carried out in [4]–[7]. Another interesting topic is how to ameliorate the network performance by minimizing the operational cost (see [8], [9]) or the transmission bandwidth (see [10]).

Since SDNs become more and more popular nowadays, the network controllers involve more and more switches to manage the networks. Although SDNs facilitate management-flexibility for these controllers, the constant growth of the flows leads to long response time (lag) and it will cause serious traffic jam. To decrease the response time, efforts can be found in improvements of algorithms (see [11], [12]) or better control of the flows (see [13], [14]). Because the flows keep increasing in networks, the latency and processing-delay, which emerge from frequent updates of the data plane, are tackled in [15]–[19] while the transmission security across each plane and link is studied and strengthened in [20]–[23].

Quite often, information fusion (multiple data-streams, which are generated by different sources, nodes, or functions, need to be merged into a single flow for transmission) is required to take place. In order to merge and recover multiple data-streams successfully and efficiently, sophisticated

Manuscript received February 16, 2018; revised June 1, 2019; accepted October 21, 2019; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor E. Yeh. Date of publication November 15, 2019; date of current version December 17, 2019. This work was supported in part by the Ministry of Science and Technology of Taiwan under Project MOST 108-2221-E-007-019. *(Corresponding author: Hsiao-Chun Wu.)*

E. Y.-N. Sun and S. C.-H. Huang are with the Department of Electrical Engineering, National Tsing Hua University, Hsinchu 30013, Taiwan (e-mail: s103064533@m103.nthu.edu.tw; chhuang@ee.nthu.edu.tw).

H.-C. Wu is with the School of Electrical Engineering and Computer Science, Louisiana State University, Baton Rouge, LA 70803 USA (e-mail: wu@ece.lsu.edu).

This article has supplementary downloadable material available at http://ieeexplore.ieee.org, provided by the authors.

Digital Object Identifier 10.1109/TNET.2019.2949823

designs are necessary in SDNs. In the existing packet-based networks, the functions related to information fusion have to be specified in the pre-defined packet fields. However, in the SDNs (such as OpenFlow), fast packet classification is quite challenging especially when the number of data-streams can change over time. With the increasing network traffic, fast search schemes for packet-headers are proposed with low computational-burden (see [24]) and dynamic memory-usage (see [25], [26]). To mitigate the network traffic load, the number of packets for transmission is reduced in [27] and [28]. Moreover, the transmission bandwidth is reduced by multiplexing different flows and compressing packet-headers in [28].

The aforementioned prevalent network operations using forwarding table management and packet based transmission are very inconvenient for information fusion of data-streams generated by ad hoc sources. We attempt to go beyond the fixed packet structure to design a dynamic, flexible, and scalable information-fusion framework, called "software-defined multiplexing code" (SDMC). The preliminary results are reported in [29]. This new technique can be deemed a *data-stream multiplexing* approach. The existing multiplexing methods, such as time-, frequency-, code-, and space-division multiplexers, have to rely on evenly-shared physical resources. When multiple data-streams have different information rates, these evenly-sharing approaches are not appropriate in practice. The SDMC can easily accommodate the information-rate discrepancy among multiple data-streams (users) and does not rely on any packet-based network. In this paper, we provide a thorough theoretical analysis and evaluation of SDMC and performance comparison between our proposed SDMC and the conventional packet-based scheme for information fusion. The major contribution of this work can be highlighted as follows:

- We propose a novel "*packetless*" codec (coder/decoder) which can multiplex and demultiplex data-streams with different data-rates simply by use of *delimiters* (separators of different data streams) and *parenthetical insertions* (payloads of data streams).
- We provide a fast systematic delimiter design procedure, which can lead to any number of delimiters for any number of data-streams (users).
- Three different kinds of SDMC schemes, namely *distributed SDMC* (or D-SDMC), *hierarchical SDMC* (or H-SDMC), and *hybrid SDMC* (or D/H-SDMC), are proposed here.
- Two crucial information-fusion performance measures, namely *coding efficiency* and *data-transmission intermittency*, are defined and formulated for all three above-mentioned SDMC schemes theoretically. Several examples are presented to compare these three SDMC schemes and the conventional packet-based method in terms of coding efficiency and data-transmission intermittency.
- The optimal parenthetical insertion strategy is derived in the sense of minimum data-transmission intermittency for three SDMC schemes.

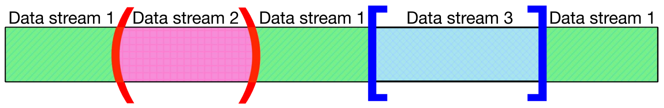


Fig. 1. Illustration of the D-SDMC data structure.

Nomenclature: $\mathcal{Z}^+$ denotes the set of positive integers. $\mathbb{E}[\ ]$ stands for statistical expectation.

II. Introduction of Software-Defined Multiplexing Code

A. Basic SDMC Structure

In a packetless network, the data constitute a continuous symbol stream (bits) without clear boundaries to indicate any packet start or end. If only a single transmission medium is available to convey different data streams which are generated by different sources, a packetless network cannot handle such multi-data-stream (or multi-user) transmission. Therefore, we propose a new *multiplexing/demultiplexing* scheme, a.k.a. SDMC for combining (merging) different data streams to transmit altogether through a medium [29]. Take a simple example for illustration here. Three different data streams can be found in Figure 1. One may choose one of them (usually with the highest data-rate) as the "main payload" while the others serve as "parenthetical insertions". Each data stream should be separated from the main payload or other data stream(s) by the 'delimiters'" (illustrated by red parentheses and blue brackets in Figure 1) to distinguish any two data streams next to each other. Note that the parenthetical insertion can be carried out anywhere in the main payload and thus it can remove the packet limitation to achieve tremendous flexibility beyond any *optimal packet design* scheme [30]–[33]. In general, a packet consists of control information (including packet-headers and trailers) and user data (packet-body or payload). As part of the required control information, the protocol needs to preserve the specific field(s) (in a packet-header) for counting how many different packets are accommodated in a multiplexed data-stream. Therefore, it is really inflexible. However, our propsed SDMC scheme relaxes this packet-format limitation and is more flexible for information fusion (multiplexing). A delimiter can be split into two parts: a *pre-delimiter* consisting of d symbols, say $(p_1, p_2, \ldots, p_d)$, and another symbol (optional), say p_{d+1}, if one needs to specify the left or right delimiter; therefore, a delimiter contains $d+1$ symbols totally. The first $d-1$ symbols $(p_1, p_2, \ldots, p_{d-1})$ in a delimiter is called a *check string*. If such a check string occurs in the original payload, one needs to *stuff* the payload by inserting an additional symbol $p_{\mathbf{S}}$ immediately after this check string (in the payload) such that $\cdots, p_1, p_2, \cdots, p_{d-1}, \underset{\substack{\uparrow \\ \text{stuffed symbol}}}{p_{\mathbf{S}}}, \cdots$ and $p_{\mathbf{S}} \neq p_d$. This operation is called "*stuffing*". After the combined (mulitplexed) sequence is received, the corresponding "unstuffing" operation is necessary, which will identify whether a check string belongs to a delimiter or a *stuffed payload*. Refer to [29] for details and examples. Algorithm 1 briefly outlines stuffing and unstuffing algorithms.

Algorithm 1 Stuffing/Unstuffing Algorithm

```
**Stuffing:**
1: if $(p_1, p_2, \cdots, p_{d-1})$ occurs in the original payload then
2:     Stuff (insert an additional symbol) $p_{\mathbf{S}}$ right after $p_{d-1}$,
       where $p_{\mathbf{S}} \neq p_d$.
3: end if

**Unstuffing:**
4: if $(p_1, p_2, \cdots, p_{d-1})$ occurs in the stuffed payload then
5:     Unstuff (remove) the symbol $p_{\mathbf{S}}$ right after $p_{d-1}$.
6: end if
```

B. Delimiter Construction

Delimiter design for SDMC is still an open problem. The key issue in this design is to determine the check string which can be searched *fast* during stuffing and unstuffing. As we propose in the previous paper [29], one may invoke the Knuth-Morris-Pratt (KMP) algorithm to search the check string in a fast manner. The complexity of this search depends on the *failure function* f, which is defined by

$$f(j) \overset{\text{def}}{=} \max\left\{k < j \,|\, p_\ell = p_{j-k+\ell},\ \ell = 1, 2, \ldots, k\right\} - 1, \quad (1)$$

if such $k \in \mathcal{Z}^+$ exists. Otherwise, $f(j) \overset{\text{def}}{=} -1$. Therefore, the failure function $f(j)$ is symbol-dependent (on the symbol index j). Given a fixed check-string length, the check string with $f(j) = -1, \forall j$ can lead to the "*fastest*" search result [29]. Let $\boldsymbol{p}(j) \overset{\text{def}}{=} (p_1, p_2, \ldots, p_j)$ be a string with length j. One may extract a substring from $\boldsymbol{p}(j)$ such that $\boldsymbol{p}(j : \mu \to \nu) \overset{\text{def}}{=} (p_\mu, p_{\mu+1}, \cdots, p_\nu)$, where $1 \leq \mu \leq \nu \leq j$. For notational convenience, we further define $\boldsymbol{p}^{\succ}(k|j) \overset{\text{def}}{=} \boldsymbol{p}(j : 1 \to k)$ and $\boldsymbol{p}^{\prec}(k|j) \overset{\text{def}}{=} \boldsymbol{p}(j : j-k+1 \to j)$.

The "*best*" (*fastest* to be searched) pre-delimiter is $\boldsymbol{p}(d) = (p_1, p_2, \ldots, p_d)$ which satisfies $f(j) = -1$, where $1 \leq j \leq d$. The "best" pre-delimiter actually means the pre-delimiter which can be *searched fastest*. In the SDMC codec, we propose to adopt the Knuth-Morris-Pratt (KMP) algorithm to search the check string. In the KMP algorithm, there are two operations. One is the failure-function computation while the other is the matching operation [34]. The time-complexity for a failure-function computation is $\mathcal{O}(d)$ with respect to the length-d string and the time-complexity for a matching operation throughout the length-ℓ sequence is $\mathcal{O}(\ell)$. Hence the total time-complexity of the KMP algorithm for a length-d string (for example, pre-delimiter) search is $\mathcal{O}(\ell + d)$. Obviously, the matching operation is necessary, so the corresponding time-complexity $\mathcal{O}(\ell)$ is fixed. If $f(j) = -1$ is satisfied for all pre-delimiters the SDMC adopts, the failure function has a constant value and hence the time-complexity $\mathcal{O}(d)$ is not needed. That is, one doesn't need to compute the failure-function anymore. Hence, the total time-complexity becomes $\mathcal{O}(\ell)$ and thus this kind of pre-delimiters can be searched fastest among all pre-delimiters of the same string length.

Choose three distinct symbols $p_{\mathbf{L}}$, $p_{\mathbf{R}}$, $p_{\mathbf{S}}$ such that the left delimiter is $\boldsymbol{p}_{\mathbf{L}}(d+1) \overset{\text{def}}{=} (p_1, p_2, \ldots, p_d, p_{\mathbf{L}})$, the right delimiter is $\boldsymbol{p}_{\mathbf{R}}(d+1) \overset{\text{def}}{=} (p_1, p_2, \ldots, p_d, p_{\mathbf{R}})$, and $p_{\mathbf{S}}$ is the stuffed symbol. We further denote $\boldsymbol{p}_{\mathbf{L}}(d+1)$ and $\boldsymbol{p}_{\mathbf{R}}(d+1)$ by "lDLM" and "rDLM", respectively. If one cannot obtain three distinct symbols, choose $p_{\mathbf{S}} = p_1$. The following corollaries and theorems are presented to guide how to design "fast searchable" delimiters and enumerate the capacity of these delimiters.

Corollary 1: For $\kappa_j \in \mathcal{Z}^+$, $f(j) = \kappa_j - 1$ if and only if $k = \kappa_j$ is the largest integer such that $\boldsymbol{p}^{\succ}(k|j) = \boldsymbol{p}^{\prec}(k|j)$.

Corollary 2: We always have $f(j) < j - 1, \forall j$.

Corollary 3: If $f(j) = \kappa_j - 1$, where $\kappa_j \in \mathcal{Z}^+$, the following statement is true:

$$p_j = p_{\kappa_j} = p_{f(j)+1}. \quad (2)$$

Theorem 1: If $f(j) = \kappa_j - 1$, where $\kappa_j \in \mathcal{Z}^+$, the following statement is true:

$$p_j = p_{f^{(i)}(j)+1}, \quad \text{for} \quad i = 1, 2, \ldots, m-1, \quad (3)$$

where $f^{(1)}(j) \overset{\text{def}}{=} f(j)$, $f^{(i)}(j) \overset{\text{def}}{=} f(f^{(i-1)}(j) + 1)$, and $f^{(m)}(j) = -1$. Note that m depends on j.

Proof: See the appendix in the supplementary document. ■

Corollary 4: If $f(j) = \kappa_j - 1$, where $\kappa_j \in \mathcal{Z}^+$, according to Corollary 1 and Theorem 1, one can get

$$\boldsymbol{p}^{\succ}\big(f^{(i)}(j) + 1|j\big) = \boldsymbol{p}^{\prec}\big(f^{(i)}(j) + 1|j\big), \quad (4)$$

where $1 \leq i \leq m - 1$.

Theorem 2: If $m_1 < m_2$, one may follow Theorem 1 to obtain

$$f^{(m_1)}(j) > f^{(m_2)}(j). \quad (5)$$

Moreover, $\boldsymbol{p}^{\succ}\big(f^{(m_2)}(j) + 1|j\big)$ is a substring of $\boldsymbol{p}^{\prec}\big(f^{(m_1)}(j) + 1|j\big)$.

Proof: See the appendix in the supplementary document. ■

Theorem 3: For all j, $\boldsymbol{p}^{\succ}\big(k|j\big) = \boldsymbol{p}^{\prec}\big(k|j\big)$ if and only if $k = f^{(i)}(j) + 1$, where $1 \leq i \leq m - 1$.

Proof: See the appendix in the supplementary document. ■

Corollary 5: By Theorems 1 and 3, we can rewrite the failure function f as

$$f(j) = \begin{cases} f^{(m)}(j-1) + 1, & \text{if } p_{[f^{(m)}(j-1)+1]+1} = p_j, \\ & \text{choose the smallest } m, \\ -1, & \text{otherwise.} \end{cases} \quad (6)$$

Theorem 4: For $1 \leq j \leq d$, $f(j) = -1$ if and only if $p_1 \neq p_j$, for $2 \leq j \leq d$.

Proof: See the appendix in the supplementary document. ■

Definition 1: The set (collection) of all pre-delimiters satisfying Theorem 4 is expressed by

$$\mathcal{P}_d^q \overset{\text{def}}{=} \big\{\boldsymbol{p}(d) = (p_1, p_2, \ldots, p_d) \mid p_1 \neq p_j, \quad \forall 2 \leq j \leq d\big\}, \quad (7)$$

where q is the modulation order and d is the length of a pre-delimiter. The *cardinality* of $\mathcal{P}_d^q$ is denoted by $|\mathcal{P}_d^q|$.

Theorem 5: If $f(j) = -1$, for $1 \leq j \leq d$, $|\mathcal{P}_d^q| = q(q-1)^{d-1}$ (it implies that there are totally $q(q-1)^{d-1}$ different pre-delimiters), where q is the modulation order and d is the length of a pre-delimiter.

TABLE I
BINARY FAST-SEARCHABLE PRE-DELIMITERS OF MODULATION ORDER $q = 2$

	$q = 2$	
	$p_1 = 0$	$p_1 = 1$
$d = 2$	01	10
$d = 3$	011	100
$d = 4$	0111	1000
$d = 10$	0111111111	1000000000
$d = 15$	011111111111111	100000000000000

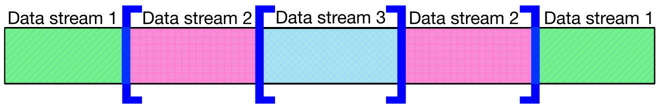


Fig. 2. Illustration of the H-SDMC data structure.

Proof: See the appendix in the supplementary document. ■

Corollary 6: For $q = 2$ (binary data-stream), $|\mathcal{P}_d^2| = 2$, $\forall d \in \mathcal{Z}^+$. There are only two different pre-delimiters whose failure functions satisfy $f(j) = -1$, $\forall j$. They are $\underbrace{011\cdots1}_{d \text{ bits}}$ and $\underbrace{100\cdots0}_{d \text{ bits}}$.

Typical examples of fast-searchable pre-delimiters can be found in Table I (for binary symbols) according to Corollary 6 and Table II (for quadrature phase-shift keying or QPSK symbols) according to Theorem 4.

III. STRUCTURES AND ANALYSES OF D-SDMC, H-SDMC, AND D/H-SDMC

According to [29], three different kinds of SDMC schemes can be used, namely *distributed SDMC* (D-SDMC), *hierarchical SDMC* (H-SDMC), and *hybrid SDMC* (D/H-SDMC). Their structures and performance analyses on *coding efficiency* and *data-transmission intermittency* will be presented in this section.

Definition 2: ***Distributed SDMC*** (D-SDMC) is a multiplexing scheme that different data streams will be inserted into the main payload (the data stream with the highest data rate) in separate sections periodically (say per session), as illustrated in Figure 1.

The encoding and decoding procedures of D-SDMC are presented in Algorithm 2. Suppose that Data-stream 1 is the main payload and $n-1$ other data streams will be inserted into the main payload, where i denotes the data stream index.

Definition 3: ***Hierarchial SDMC*** (H-SDMC) is a multiplexing scheme that different data streams will be inserted into the main payload (the data stream with the highest data rate) in a nested structure as illustrated in Figure 2.

The encoding and decoding procedures of the H-SDMC approach are presented in Algorithm 3. Suppose that Data-stream 1 is the main payload and $n-1$ other data streams will be inserted into the main payload, where i denotes the data stream index.

Definition 4: ***Hybrid SDMC*** (D/H-SDMC) is a *hybrid* multiplexing scheme combining both D-SDMC and H-SDMC.

Algorithm 2 D-SDMC

```
**Encoding**
1:  for ($i \leftarrow 2$; $i \le n$; $i \leftarrow i+1$) do
2:      Generate a new pre-delimiter $\boldsymbol{p}_i(d)$ for Data-stream $i$.
3:      Stuffed Data-stream $i \leftarrow$ Data-stream $i$
4:      for ($j \leftarrow 2$; $j \le i$; $j \leftarrow j+1$) do
5:          Stuffed Data-stream $i \leftarrow$ Apply the stuffing algorithm
            again to Stuffed Data-stream $i$. The stuffing operation
            here is with respect to $\boldsymbol{p}_j(d)$.
6:      end for
7:      Apply the stuffing algorithm to the (stuffed) main
        payload. The stuffing operation here is with respect
        to $\boldsymbol{p}_i(d)$.
8:      Find a position $x$ to avoid the conflict with previously
        inserted data stream(s) such that the beginning, say $x$,
        of any inserted data stream can never occur within
        another inserted data stream. Insert the concatenated
        stream (lDLM: Stuffed Data-stream $i$: rDLM) at $x$.
9:  end for

**Decoding**
10: for ($i \leftarrow n$; $i \ge 2$; $i \leftarrow i-1$) do
11:     Search the entire data stream for the corresponding
        pre-delimiter $\boldsymbol{p}_i(d)$.
12:     Extract Stuffed Data-stream $i$.
13:     Apply the unstuffing algorithm for Stuffed Data-stream
        $i$ and the stuffed main payload. The unstuffing
        operation here is with respect to $\boldsymbol{p}_i(d)$.
14: end for
```

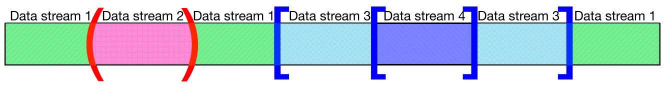


Fig. 3. Illustration of the D/H-SDMC data structure.

Hence the number of pre-delimiters is smaller than $n-1$ and larger than 1, where n is the number of data streams as illustrated in Figure 3.

Definition 5: If a given pre-delimiter is a substring of another pre-delmiter, it will cause "*conflict*". As illustrated in Table III, the parenthetical-insertions containing Data-streams i, or PI_i, $i = 1, 2$ are separated by the corresponding delimiters. It is clear that the SDMC decoder cannot uniquely decode this data stream due to the conflict in delimiters.

According to Theorem 5, when the number of data streams n gets large, one cannot find enough delimiters for a fixed modulation order q and a fixed pre-delimiter length d. It is likely that we need to use the delimiters of different lengths (modulation orders) to accommodate this situation. Thus, we need to choose those variable-length (variable-modulation-order) delimiters not in conflict. The following theorems are presented for this purpose.

Theorem 6: For the pre-delimiters of modulation-order q and length d_1, if all pre-delimiters in $\mathcal{P}_{d_1}^q$ are used, they will cause conflict with all pre-delimiters in $\mathcal{P}_{d_2}^q$, where $d_2 \ne d_1$.

Proof: See the appendix in the supplementary document. ■

TABLE II

PRE-DELIMITERS OF MODULATION ORDER $q = 4$, WHERE $\mathrm{j} \overset{\text{DEF}}{=} \sqrt{-1}$

	$q = 4$			
	$p_1 = -1 - \mathrm{j}$	$p_1 = -1 + \mathrm{j}$	$p_1 = 1 - \mathrm{j}$	$p_1 = 1 + \mathrm{j}$
$d = 2$	$(-1-\mathrm{j},-1+\mathrm{j})$	$(-1+\mathrm{j},-1-\mathrm{j})$	$(1-\mathrm{j},-1-\mathrm{j})$	$(1+\mathrm{j},-1-\mathrm{j})$
	$(-1-\mathrm{j},1-\mathrm{j})$	$(-1+\mathrm{j},1-\mathrm{j})$	$(1-\mathrm{j},-1+\mathrm{j})$	$(1+\mathrm{j},-1+\mathrm{j})$
	$(-1-\mathrm{j},1+\mathrm{j})$	$(-1+\mathrm{j},1+\mathrm{j})$	$(1-\mathrm{j},1+\mathrm{j})$	$(1+\mathrm{j},1-\mathrm{j})$
$d = 3$	$(-1-\mathrm{j},-1+\mathrm{j},-1+\mathrm{j})$	$(-1+\mathrm{j},-1-\mathrm{j},-1-\mathrm{j})$	$(1-\mathrm{j},-1-\mathrm{j},-1-\mathrm{j})$	$(1+\mathrm{j},-1-\mathrm{j},-1-\mathrm{j})$
	$(-1-\mathrm{j},-1+\mathrm{j},1-\mathrm{j})$	$(-1+\mathrm{j},-1-\mathrm{j},1-\mathrm{j})$	$(1-\mathrm{j},-1-\mathrm{j},-1+\mathrm{j})$	$(1+\mathrm{j},-1-\mathrm{j},-1+\mathrm{j})$
	$(-1-\mathrm{j},-1+\mathrm{j},1+\mathrm{j})$	$(-1+\mathrm{j},-1-\mathrm{j},1+\mathrm{j})$	$(1-\mathrm{j},-1-\mathrm{j},1+\mathrm{j})$	$(1+\mathrm{j},-1-\mathrm{j},1-\mathrm{j})$
	$(-1-\mathrm{j},1-\mathrm{j},-1+\mathrm{j})$	$(-1+\mathrm{j},1-\mathrm{j},-1-\mathrm{j})$	$(1-\mathrm{j},-1+\mathrm{j},-1-\mathrm{j})$	$(1+\mathrm{j},-1+\mathrm{j},-1-\mathrm{j})$
	$(-1-\mathrm{j}\ 1-\mathrm{j},1-\mathrm{j})$	$(-1+\mathrm{j},1-\mathrm{j},1-\mathrm{j})$	$(1-\mathrm{j},-1+\mathrm{j},-1+\mathrm{j})$	$(1+\mathrm{j},-1+\mathrm{j},-1+\mathrm{j})$
	$(-1-\mathrm{j},1-\mathrm{j},1+\mathrm{j})$	$(-1+\mathrm{j},1-\mathrm{j},1+\mathrm{j})$	$(1-\mathrm{j},-1+\mathrm{j},1+\mathrm{j})$	$(1+\mathrm{j},-1+\mathrm{j},1-\mathrm{j})$
	$(-1-\mathrm{j},1+\mathrm{j},-1+\mathrm{j})$	$(-1+\mathrm{j},1+\mathrm{j},-1-\mathrm{j})$	$(1-\mathrm{j},1+\mathrm{j},-1-\mathrm{j})$	$(1+\mathrm{j},1-\mathrm{j},-1-\mathrm{j})$
	$(-1-\mathrm{j},1+\mathrm{j},1-\mathrm{j})$	$(-1+\mathrm{j},1+\mathrm{j},1-\mathrm{j})$	$(1-\mathrm{j},1+\mathrm{j},-1+\mathrm{j})$	$(1+\mathrm{j},1-\mathrm{j},-1+\mathrm{j})$
	$(-1-\mathrm{j},1+\mathrm{j},1+\mathrm{j})$	$(-1+\mathrm{j},1+\mathrm{j},1+\mathrm{j})$	$(1-\mathrm{j},1+\mathrm{j},1+\mathrm{j})$	$(1+\mathrm{j},1-\mathrm{j},1-\mathrm{j})$

TABLE III

ILLUSTRATION OF THE CONFLICT BETWEEN THE DELIMITERS CORRESPONDING TO TWO DATA STREAMS

Payload	lDLM$_1$	PI$_1$	rDLM$_1$	Payload	lDLM$_2$	PI$_2$	rDLM$_2$	Payload
-	01111	-	01110	-	0111111	-	0111110	-

Algorithm 3 H-SDMC

```
    Encoding
 1: Generate a pre-delimiter $\boldsymbol{p}(d)$.
 2: Apply the stuffing algorithm to Data-stream 1.
 3: for $(i \leftarrow 2;\ i \le n;\ i \leftarrow i+1)$ do
 4:     Apply the stuffing algorithm to Data-stream $i$.
 5:     Find a position $x$ within the stuffed data stream $i-1$
        to insert the stuffed Data-stream $i$.
 6:     Insert the concatenated stream (lDLM:stuffed
        Data-stream $i$:rDLM) at $x$ such that the beginning, say
        $x$, and the end of the concatenated stream are both
        located within the stuffed data stream $i-1$.
 7: end for

    Decoding
 8: for $(i \leftarrow n;\ i \ge 2;\ i \leftarrow i-1)$ do
 9:     Search the entire data stream for the innermost
        pre-delimiter $\boldsymbol{p}(d)$.
10:     Extract the stuffed Data-stream $i$.
11:     Apply the unstuffing algorithm to recover the stuffed
        Data-stream $i$.
12: end for
13: Apply the unstuffing algorithm to recover the stuffed Data-
    stream 1.
```

According to Theorem 5, one may easily conclude that the larger q, the more pre-delimiters one can obtain for a fixed d. If one just changes the modulation order from q to q' and then remodulate everything (data and delimiters), will the number of pre-delimiters still increase? First, enlarge the modulation order from q to $q' = q^k$, where $k \in \mathcal{Z}^+$. Second, the length of the pre-delimiter, d, should be changed to $d' \in \mathcal{Z}^+$, where $d' = d/\left\lceil \log_q(q') \right\rceil = d/k \ge 2$, after the modulation order changes. Since q, q', d, and d' are all integers, k must be an integer as well. The length of the original payload data, l, is changed to l' after remodulation. Note that the original pre-delimiter length d should be a multiple of k. Theorems 7 and 8 discuss the important properties pertinent to the aforementioned modulation-order change.

Theorem 7: If one changes the modulation-order from q to $q' = q^k$ and the length of the pre-delimiter from d to $d' = d/k$, the following will be true:

$$|\mathcal{P}_d^q| < |\mathcal{P}_{d'}^{q'}|. \tag{8}$$

Proof: See the appendix in the supplementary document. ■

Theorem 8: Suppose $\boldsymbol{p}(d) \in \mathcal{P}_d^q$. When the modulation order is changed from q to q' and the length of the pre-delimiters from d to d', any length-d pre-delimiter $\boldsymbol{p}(d)$ can be remodulated to be a length-d' pre-delimiter $\boldsymbol{p}(d')$. Then,

$$\boldsymbol{p}(d') \in \mathcal{P}_{d'}^{q'}. \tag{9}$$

Proof: See the appendix in the supplementary document. ■

For example, for $q = 2$ and $d = 8$, by Corollary 6, there are only two viable pre-delimiters: "01111111" and "10000000", both of which belong to $\mathcal{P}_8^2$. If we change $q = 2$ to $q' = q^k = 2^4 = 16$ and the length from $d = 8$ to $d' = d/k = 8/4 = 2$, we can obtain "$01111111''$ = "0×7f" and "$10000000''$ = "$0 {\times} 80$". Obviously, both "$0 {\times} 7$f" and "0×80" belong to $\mathcal{P}_2^{16}$.

By Theorems 7 and 8, the modulation-order change from q to q' and the pre-delimiter-length change from d to d' can expand the capacity of pre-delimiters while the pre-delimiters in $\mathcal{P}_d^q$ are all contained in $\mathcal{P}_{d'}^{q'}$. Although this transformation can create more viable pre-delimiters, extra resource in hardware and computation is needed for the accommodation. To overcome this drawback, an alternative approach to create more delimiters will be presented in the next section.

IV. DELIMITERS BASED ON FAST-SEARCHABLE STRINGS AND SEMI-FAST-SEARCHABLE STRINGS

The previous sections address the viable (fast-searchable) delimiters and they are limited in capacity as discussed in Section III. In this section, we would like to relax this restriction to increase the delimiter capacity without any need of modulation-order increase. Let's start with the definition of *fast-searchable string*.

Definition 6: A string (sequence) is represented by $\boldsymbol{p}(d) \overset{\text{def}}{=} (p_1, p_2, \cdots, p_d)$. If the corresponding failure function $f(j) = -1$, $j = 1, 2, \cdots, d$, $\boldsymbol{p}(d)$ is a fast-searchable string (FSS).

Let's focus on searching (spotting) length-d string(s) over a length-l sequence. In the KMP algorithm, the time-complexity for failure-function computations is $\mathcal{O}(d)$ with respect to the length-d string and the time-complexity for matching operations throughout the length-l sequence is $\mathcal{O}(l)$. Hence the total time-complexity of the KMP algorithm in a length-d string (for example, pre-delimiter) search is $\mathcal{O}(l + d)$. Note that the conventional *template-matching* method has the time-complexity $\mathcal{O}(dl)$ to serve on the same purpose.

Corollary 7: For any FSS in compliance with Definition 6, the time-complexity for spotting such a string is $\mathcal{O}(l)$, where l denotes the entire sequence length.

Another way to solve the problem of the limited pre-delimiters of modulation order q and the problem of the extension of pre-delimiters is to relax the failure function requirement. In the expanded pre-delimiter set, we have to match the last symbol with the first symbol in any pre-delimiter. This expanded set can be manifested in the following definition.

Definition 7: A string (sequence) is represented by $\boldsymbol{p}(d) \overset{\text{def}}{=} (p_1, p_2, \cdots, p_d)$. If $d = d_1 + d_2$ and $d_1, d_2 \in \mathcal{Z}^+$, the corresponding failure function is given by Eq. (10), as shown at the bottom of this page. When $d_1 > d_2$, $\boldsymbol{p}(d)$ is a *semi-fast-searchable string* (SFSS).

Corollary 8: According to Theorem 4, a string $\boldsymbol{p}(d) \overset{\text{def}}{=} (p_1, p_2, \cdots, p_d)$ is an SFSS if and only if $p_1 \neq p_j$, $2 \leq j \leq d_1$, and $p_d = p_1$.

Theorem 9: For any SFSS in compliance with Definition 7, the time-complexity for spotting such a string is $\mathcal{O}(l + d_2)$, where l denotes the entire sequence length.

Proof: See the appendix in the supplementary document. ■

Definition 8: The set (collection) of the pre-delimiters is expressed by Eq. (11), as shown at the bottom of this page, where q is the modulation order and d is the pre-delimiter length. These delimiters are all SFSSs according to Corollary 8. According to Definition 1, $\mathcal{P}^q_{d:0} = \mathcal{P}^q_d$.

Theorem 10: For the SFSS pre-delimiters, $|\mathcal{P}^q_{d_1:d_2}| = q^{d_2}(q-1)^{d_1-1}$, where q is the modulation order and $d_1 + d_2$ is the pre-delimiter length.

Proof: See the appendix in the supplementary document. ■

Corollary 9: For $q = 2$, we can have $|\mathcal{P}^2_{d_1:d_2}| = 2^{d_2}$.

Theorem 11: For the SFSS pre-delimiters, if each of them starts with the same symbol p_1, every substring to be stuffed with respect to the current pre-delimiter will not contain the previously stuffed symbol(s).

Proof: See the appendix in the supplementary document. ■

Corollary 10: By Theorem 11, if choosing all SFSSs starting with the same symbol p_1 from $\mathcal{P}^q_{d_1:d_2}$, one can have $q^{d_2-1}(q-1)^{d_1-1}$ pre-delimiters.

Theorem 12: For the SFSS pre-delimiters $\mathcal{P}^q_{d_1:d_2}$ and $q > 2$, if one chooses a stuffed symbol p_{S} where $p_1 \neq p_{\text{S}}$, then every substring to be stuffed with respect to the current pre-delimiter will not contain the previously stuffed symbol(s). Moreover, there are $q^{d_2-1}(q-1)^{d_1}$ pre-delimiters.

Proof: See the appendix in the supplementary document. ■

Definition 9: If a collection of SFSS pre-delimiters is denoted by $\mathcal{P}^q_{d_1:d_2}$, its subset of pre-delimiters, each of which starts with the same symbol p_1, is denoted by $\acute{\mathcal{P}}^q_{d_1:d_2}$; $\mathcal{P}^q_{d_1:d_2}$'s subset of pre-delimiters, each of which starts with a different symbol from any stuffed symbol, is denoted by $\grave{\mathcal{P}}^q_{d_1:d_2}$.

For those pre-delimiters satisfying Theorem 11, stuffed symbols are independent. There is no relationship between any two stuffed symbols. This desirable property may benefit the decoding flexibility of Algorithm 2. In Algorithm 2, The previous stuffed symbol(s) may affect the subsequent stuffed symbol(s) within the stuffed payload. For example, if there are two SFSS pre-delimiters: $\boldsymbol{p}_1(7)$ = "0111000" and $\boldsymbol{p}_2(7)$ ="1000111", and the payload data stream is "011100000111", the stuffed-payload becomes "011100-**1**-00011-**0**-1". The previously stuffed symbol "**1**" will affect the subsequent stuffed symbol "**0**". Hence the decoding procedure must follow the strict unstuffing order. However, Theorem 11 or Theorem 12 can be utilized to relax this unfavorable restriction. Moreover, one can take an arbitrary unstuffing order across data streams by Theorem 11 or Theorem 12.

V. Expected Numbers of Stuffed Symbols

All pre-delimiters discussed in Sections V-VII belong to either $\acute{\mathcal{P}}^q_{d_1:d-d_1}$ or $\grave{\mathcal{P}}^q_{d_1:d-d_1}$. In fact, the number of stuffed symbols coincides with the number of pre-delimiter's occurrences in the original payload. Consider a payload length l and a pre-delimiter length d. Let q be the modulation order and $g \overset{\text{def}}{=} \lfloor l/(d-1) \rfloor$, where $\lfloor\ \rfloor$ is the *integer rounding-down operator.* Assume that the symbols in a payload sequence are i.i.d. (identically and independently distributed) and each symbol has a uniform distribution. We define S_e and S_e^* as follows: S_e specifies the total number of length-d pre-delimiter's occurrences over all length-l payload sequences in each of

$$f(j) = \begin{cases} -1, & \text{if } 1 \leq j \leq d_1, \\ \max\left\{k < j | p_\ell = p_{j-k+\ell},\ \ell = 1, 2, ..., k\right\} - 1, & \text{if } d_1 + 1 \leq j \leq d_1 + d_2 - 1, \\ 0, & \text{if } j = d. \end{cases} \tag{10}$$

$$\mathcal{P}^q_{d_1:d_2} \overset{\text{def}}{=} \left\{\boldsymbol{p}(d_1 + d_2) = (p_1, \ldots, p_{d_1}, p_{d_1+1}, \ldots, p_{d_1+d_2}) \mid p_1 \neq p_j, \forall 2 \leq j \leq d_1, p_{d_1+d_2} = p_1\right\} \tag{11}$$

which the delimiter occurs exactly e-times; S_e^* specifies the total number of length-d pre-delimiter's occurrences over all length-l payload sequences in each of which e delimiters can be found at certain symbol positions. We have the following relations:

$$\begin{aligned}
\binom{l-(d-1)+1}{1}S_1^* &= \binom{1}{1}S_1+\binom{2}{1}S_2+\cdots+\binom{g}{1}S_g\\
\binom{l-2(d-1)+2}{2}S_2^* &= \binom{2}{2}S_2+\cdots+\binom{g}{2}S_g\\
&\vdots\\
\binom{l-g(d-1)+g}{g}S_g^* &= \binom{g}{g}S_g
\end{aligned} \tag{12}$$

and

$$S_e^* = q^{l-e(d-1)}. \tag{13}$$

Suppose that the number of stuffed symbols with respect to $\boldsymbol{p}_i(d)$ in a length-l payload sequence is s_i. According to Eqs. (12) and (13), when $e = 1$, the expected number of stuffed symbols with respect to $\boldsymbol{p}_i(d)$ in a length-l payload sequence is given by

$$\mathbb{E}[s_i] = \sum_{u=1}^{g}\frac{u\,S_u}{q^l} = \frac{\binom{l-(d-1)+1}{1}S_1^*}{q^l} = (l-d+2)\,q^{-(d-1)}. \tag{14}$$

After we stuff the symbols into the original payload with respect to a specific pre-delimiter, it becomes the "stuffed payload" and Eq. (14) cannot be applied. For any pre-delimiter in $\acute{\mathcal{P}}^q_{d_1:d-d_1}$ or $\grave{\mathcal{P}}^q_{d_1:d-d_1}$, any stuffed symbol will not overlap with other pre-delimiters. Hence we may follow Corollary 11 to find the expected number of stuffed symbols with respect to $\boldsymbol{p}_i(d)$ for the (stuffed) payload.

Corollary 11: If a payload has been stuffed with respect to m pre-delimiters, the expected number of stuffed symbols with respect to another, say the $(m+1)^{\text{th}}$ pre-delimiter, in the stuffed payload is given by Eq. (15) shown at the bottom of this page.

The expected number of stuffed symbols with respect to all m pre-delimiters in a stuffed payload is denoted by $\mathcal{E}(l : q, d, m)$ where q is the modulation order, l is the original payload-length, and d is the pre-delimiter length. By Corollary 11, one can get Eq. (16) shown at the bottom of this page.

Corollary 12: According to Eq. (16) on next page, one can find $1-(d-1)q^{-(d-1)} > 1$. Hence

$$\begin{aligned}
\mathcal{E}(l : q, d, m) &\le \sum_{w=0}^{m-1}(l-d+2)q^{-(d-1)}\\
&= m(l-d+2)q^{-(d-1)}.
\end{aligned} \tag{17}$$

Definition 10: If there are n data streams, the expected number of stuffed symbols with respect to all m pre-delimiters in all n stuffed payloads is defined by $\mathcal{E}(l_1, ..., l_n : q, d, n, m)$. That is,

$$\mathcal{E}(l_1, ..., l_n : q, d, n, m) = \sum_{u=1}^{n}\mathcal{E}(l_u : q, d, m), \tag{18}$$

where l_u is the length of the u^{th} original payload, q is the modulation order, and d is the pre-delimiter length.

Corollary 13: According to Eq. (16), if $m_1 > m_2$, $\mathcal{E}(l : q, d, m_1) > \mathcal{E}(l : q, d, m_2)$ and $\mathcal{E}(l_1, ..., l_n : q, d, n, m_1) > \mathcal{E}(l_1, ..., l_n : q, d, n, m_2)$.

Corollary 14: By Corollary 12, one can have

$$\mathcal{E}(l_1, ..., l_n : q, d, n, m) \le m\sum_{u=1}^{n}(l_u-d+2)q^{-(d-1)}. \tag{19}$$

VI. Coding Efficiency

In this section, we would like to discuss the *coding efficiency* of SDMC and establish the theoretical performance comparison among D-SDMC, H-SDMC, and D/H-SDMC.

Definition 11: The coding efficiency of SDMC is defined by Eq. (20), as shown at the bottom of the next page, where n is the number of data streams. Note that the u^{th} stuffed-payload (after SDMC encoding) length = the length of left delimiter + the length of the u^{th} stuffed payload + the length of right delimiter.

Since the length of the stuffed payload depends on the number of stuffed symbols, the *expected coding efficiency* $\eta(l_1, ..., l_n : q, d, n, m)$ can be calculated using Eqs. (16) and (18), where l_u is the length of the u^{th} original payload, q is the modulation order, d is the pre-delimiter length, n is the number of data streams, and m is the number of pre-delimiters.

$$\begin{aligned}
\mathbb{E}[s_{m+1}] &= \left\{l-(d-1)\sum_{u=1}^{m}\mathbb{E}[s_u]-d+2\right\}q^{-(d-1)}\\
&= \left\{l-(d-1)\sum_{u=1}^{m}\left[(l-d+2)[1-(d-1)q^{(d-1)}]^{u-1}\right]q^{-(d-1)}-d+2\right\}q^{-(d-1)}
\end{aligned} \tag{15}$$

$$\begin{aligned}
\mathcal{E}(l : q, d, m) &= \sum_{w=1}^{m}\mathbb{E}[s_w] = \sum_{w=0}^{m-1}\mathbb{E}[s_{w+1}]\\
&= \sum_{w=0}^{m-1}\left\{l-(d-1)\sum_{u=0}^{w-1}\left[(l-d+2)[1-(d-1)q^{-(d-1)}]^{u}\right]q^{-(d-1)}-d+2\right\}q^{-(d-1)}
\end{aligned} \tag{16}$$

That is,

$$\eta(l_1,...,l_n:q,d,n,m) \overset{\text{def}}{=} \mathbb{E}\big[\text{coding efficiency}\big] = \frac{\sum_{u=1}^{n} l_u}{\sum_{u=1}^{n} l_u + 2(n-1)(d+1) + \mathcal{E}(l_1,...,l_n:q,d,n,m)}. \tag{21}$$

To compare the expected coding efficiencies among D-SDMC, D/H-SDMC, and H-SDMC, we fix the variables l_u, q, d, and n. Hence the only different parameter among these three SDMC schemes is m. Denote m_{H}, $m_{\text{D/H}}$, and m_{D}, which represent the numbers of pre-delimiters used in H-SDMC, D/H-SDMC, and D-SDMC, respectively. The condition to ensure $m_{\text{H}} \neq m_{\text{D/H}}$, $m_{\text{D/H}} \neq m_{\text{D}}$, and $m_{\text{H}} \neq m_{\text{D}}$ is $n \geq 4$. Thus, one has

$$n-1 = m_{\text{D}} > m_{\text{D/H}} > m_{\text{H}} = 1. \tag{22}$$

By Corollary 13 and Eq. (22), one can have

$$\begin{aligned}\mathcal{E}(l_1,...,l_n:q,d,n,m_{\text{D}}=n-1) &> \mathcal{E}(l_1,...,l_n:q,d,n,m_{\text{D/H}}) \\ &> \mathcal{E}(l_1,...,l_n:q,d,n,m_{\text{H}}=1).\end{aligned} \tag{23}$$

By Eq. (21) and Eq. (23), the respective expected coding efficiencies of D-SDMC, H-SDMC, and D/H-SDMC are related by

$$\begin{aligned}\eta(l_1,...,l_n:q,d,n,m_{\text{D}}=n-1) &< \eta(l_1,...,l_n:q,d,n,m_{\text{D/H}}) \\ &< \eta(l_1,...,l_n:q,d,n,m_{\text{H}}=1).\end{aligned} \tag{24}$$

VII. Data-Transmission Intermittency

As stated in Seciton VI, the numbers of pre-delimiters are different among D-SDMC, H-SDMC, and D/H-SDMC. This discrepancy will lead to different coding efficiencies as given by Eq. (24). In addition, the ways to multiplex (insert) data streams into the main payload are also different across the three SDMC schemes. Different multiplexing approaches will surely lead to different *data-transmission intermittencies*. The data-transmission intermittency specifies how many symbols in transmission (from other data streams) a particular data stream needs to wait until it can continue to be transmitted again. The data-transmission intermittency here is a good indicator about the *resource-sharing balance* (*fairness*). If the system spends a vast majority of time in transmitting a particular data stream but leaves other data streams idle meanwhile, such resource-sharing strategy is unfair or unbalanced. Hence, one needs to evaluate the data-transmission intermittency for checking the resource-sharing balance. Let's start from the definition of *duty cycle* for Data-stream u as follows.

Definition 12: Divide the u^{th} data stream into several sections each with equal length ξ_u, where $1 \leq u \leq n$. The duty cycle of Data-stream u is given by Eq. (25), as shown at the bottom of this page, where

$$l_u = \underbrace{\xi_u}_{\text{section length}} \times \underbrace{\varphi_u}_{\text{number of sections}}, \tag{26}$$

ξ_u, $\varphi_u \overset{\text{def}}{=} l_u/\xi_u \in \mathcal{Z}^+$, and n is the number of data streams. Note that the w^{th} stuffed-payload (after SDMC encoding) length = the length of left delimiter + the length of the w^{th} stuffed payload + the length of right delimiter.

Since the length of the stuffed payload depends on the number of stuffed symbols, the *expected duty cycle for Data-stream u*, $\rho_u(l_1,...,l_n,q,d,n,m)$, can be calculated using Eqs. (16) and (18), where l_u is the length of the u^{th} original payload, q is the modulation order, d is the pre-delimiter length, n is the number of data streams, and m is the number of pre-delimiters. That is,

$$\rho_u(l_1,...,l_n,q,d,n,m) \overset{\text{def}}{=} \mathbb{E}\big[\text{duty cycle for Data-stream } u\big] = 1 - \frac{l_u}{\sum_{w=1}^{n} l_w + 2(n-1)(d+1) + \mathcal{E}(l_1,...,l_n:q,d,n,m)}. \tag{27}$$

Definition 13: The *data-transmission intermittency for Data-stream* u is given by

$$\bar{\rho}_u(\xi_1,...,\xi_n,q,d,n,m) \overset{\text{def}}{=} \frac{\rho_u(l_1,...,l_n,q,d,n,m)}{\varphi_u}, \tag{28}$$

where ξ_u is the section length of Data-stream u, ξ_u, $\varphi_u \overset{\text{def}}{=} l_u/\xi_u \in \mathcal{Z}^+$, q is the modulation order, d is the pre-delimiter length, n is the number of data streams, and m is the number of pre-delimiters.

$$\text{coding efficiency} \overset{\text{def}}{=} \frac{\sum_{u=1}^{n} l_u}{\sum_{u=1}^{n} \text{the } u^{\text{th}} \text{ Stuffed-Payload (After SDMC Encoding) Length}} \tag{20}$$

$$\begin{aligned}\text{Duty cycle for Data-stream } u &\overset{\text{def}}{=} \frac{\sum_{w=1}^{n} \text{the } w^{\text{th}} \text{ Stuffed-Payload (After SDMC Encoding) Length} - \varphi_u\xi_u}{\sum_{w=1}^{n} \text{the } w^{\text{th}} \text{ Stuffed-Payload (After SDMC Encoding) Length}} \\ &= 1 - \frac{l_u}{\sum_{w=1}^{n} \text{the } w^{\text{th}} \text{ Stuffed-Payload (After SDMC Encoding) Length}}\end{aligned} \tag{25}$$

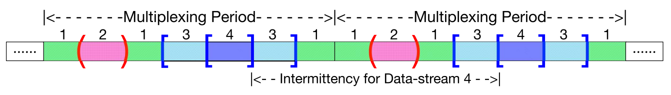

|<- - - - - - -Multiplexing Period- - - - - - ->|<- - - - - - -Multiplexing Period- - - - - - ->|
1 2 1 3 4 3 1 1 2 1 3 4 3 1
......
|<- - Intermittency for Data-stream 4 - ->|

Fig. 4. Illustration of data-transmission intermittency and multiplexing period in the D/H-SDMC scheme.

Note that the intermittency given by Eq. (28) is calculated within a *multiplexing period*, as shown in Figures 4-7. From Eqs. (27) and (28), one may simultaneously minimize the data-transmission intermittency for every data stream with respect to an optimal pre-delimiter length d. Note that all data-transmission intermittencies share the same denominator $2(n-1)(d+1)+\mathcal{E}(l_1,...,l_n : q,d,n,m)$. Therefore, by Corollary 14, we can minimize the intermittencies (for all data-streams) by

$$\min_{d}\left[2(n-1)(d+1)+m\sum_{u=1}^{n}(l_u-d+2)q^{-(d-1)}\right]. \quad (29)$$

By Corollary 10, the number of pre-delimiters, m, is restricted by the length of pre-delimiter, d. Hence one needs to determine the proper pre-delimiter length according to the following two situations:

$$\begin{cases} m > q^{\tilde{d}_2-1}(q-1)^{\tilde{d}_1-1}, & \text{Case I},\\ m \leq q^{\tilde{d}_2-1}(q-1)^{\tilde{d}_1-1}, & \text{Case II}, \end{cases} \quad (30)$$

where q is the modulation order and $\tilde{d} \overset{\text{def}}{=} \tilde{d}_1+\tilde{d}_2$ denotes the pre-delimiter length resulting from Eq. (29). The unique integer partition of $\tilde{d}$ has to satisfy both $\tilde{d}_1 > \tilde{d}_2$ and $\tilde{d}_1 - 1 \leq \tilde{d}_2 + 1$. According to Eq. (30), if $m > q^{\tilde{d}_2-1}(q-1)^{\tilde{d}_1-1}$, the number of SFSS pre-delimiters is not sufficient. Therefore, one needs to find $\ddot{d}_1$ and $\ddot{d}_2$ such that $\ddot{d}_1 \overset{\text{def}}{=} \ddot{d}_2+1$ and $\ddot{d}_2 \overset{\text{def}}{=} q^{\ddot{d}_2-1}(q-1)^{\ddot{d}_1-1} \geq m$. Henceforth, the proper pre-delimiter length is given by $d \overset{\text{def}}{=} \ddot{d}_1+\ddot{d}_2$. On the other hand, if $m \leq q^{\tilde{d}_2-1}(q-1)^{\tilde{d}_1-1}$, the proper pre-delimiter length can simply be $d \overset{\text{def}}{=} \tilde{d}$.

In the following analysis, we will derive the data-transmission intermittencies for the entire long multiplexed sequence (across different multiplexing periods).

For D/H-SDMC, as shown in Figure 4, the section-length ξ_1 will affect the intermittency of Data-stream 1. If there are m pre-delimiters, Data-stream 1 will be separated into $\varphi_1 = l_1/\xi_1$ sections. We define λ by

$$\lambda \overset{\text{def}}{=} \frac{1}{\varphi_1} = \frac{\xi_1}{l_1}, \quad \text{where } 1 \leq \varphi_1 \leq m+1. \quad (31)$$

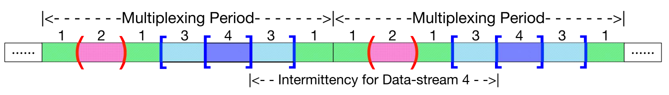
Let

$$\mathcal{L}_u \overset{\text{def}}{=} l_u + 2(d+1), \quad \text{for } u = 1,2,\ldots,n; \quad (32)$$

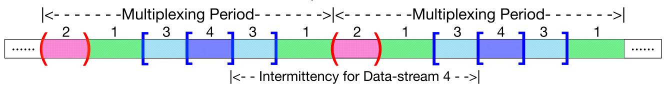
$$\mathcal{E}_{u,m} \overset{\text{def}}{=} \mathcal{E}(l_u : q,d,n,m),$$
$$\text{for } u = 1,2,\ldots,n, m = 1,2,\ldots,n-1; \quad (33)$$

and

$$\mathcal{N}(m) \overset{\text{def}}{=} \sum_{u=1}^{n} l_u + 2(n-1)(d+1) + \mathcal{E}(l_1,...,l_n : q,d,n,m), \quad \text{for } m = 1,2,\ldots,n-1. \quad (34)$$


|<- - - - - - -Multiplexing Period- - - - - - ->|<- - - - - - -Multiplexing Period- - - - - - ->|
1 2 1 3 4 3 1 1 2 1 3 4 3 1
......
|<- - Intermittency for Data-stream 4 - ->|

(a) $\varphi_1 = 3$.


|<- - - - - - -Multiplexing Period- - - - - - ->|<- - - - - - -Multiplexing Period- - - - - - ->|
2 1 3 4 3 1 2 1 3 4 3 1
......
|<- - Intermittency for Data-stream 4 - ->|

(b) $\varphi_1 = 2$.

Fig. 5. Illustration of the data-transmission intermittency in D/H-SDMC for $n = 4$ and $m = 2$.

In D/H-SDMC, apart from Data-stream 1, pick m data streams with $\xi_u = l_u$ and $n-m-1$ data streams with $\xi_u = l_u/2$. Let $\mathcal{D} \overset{\text{def}}{=} \{u|\xi_u = l_u, u \neq 1\}$, $\mathcal{H} \overset{\text{def}}{=} \{u|\xi_u = l_u/2, u \neq 1\}$, and $\mathcal{M} \overset{\text{def}}{=} \{u \neq 1|$ a section of Data-stream $\hat{u}$ is transmitted just before two consecutive sections of Data-stream 1 and all Data-stream u's are transmitted between the beginning and end of Data-stream $\hat{u}$ (including $\hat{u}$)$\}$. For example, $\mathcal{M} = \{3,4\}$ and $\mathcal{M} = \emptyset$ in Figure 5a and Figure 5b, respectively. As illustrated by Figure 5, the intermittency of Data-stream u is given by

$$\bar{\rho}_u(\xi_1,\xi_2,...,\xi_n,q,d,n,m) = \begin{cases} \dfrac{\sum_{w=1}^{n}(\mathcal{L}_w+\mathcal{E}_{w,m})-\mathcal{L}_u}{\mathcal{N}(m)}, & u \in \mathcal{D},\\ \dfrac{\sum_{w=1}^{n}(\mathcal{L}_w+\mathcal{E}_{w,m})-\mathcal{L}_u}{2\mathcal{N}(m)}, & u \in \mathcal{H},\\ \dfrac{\lambda\left[\sum_{w=1}^{n}(\mathcal{L}_w+\mathcal{E}_{w,m})+\sum_{w\in\mathcal{M}}(\mathcal{L}_w+\mathcal{E}_{w,m})-\mathcal{L}_1\right]}{\mathcal{N}(m)}, & u = 1. \end{cases} \quad (35)$$

In Eq. (35), the selection of $\mathcal{D}$ and $\mathcal{H}$ and the schedule of Data-streams 2-n will affect the data-transmission intermittency. The optimal arrangement of all data-streams arises from the minimization of the *overall data-transmission intermittency* $\Psi(\xi_1,...,\xi_n,q,d,n,m)$ which is defined by

$$\Psi(\xi_1,...,\xi_n,q,d,n,m) \overset{\text{def}}{=} \sum_{u=1}^{n}\frac{l_u\bar{\rho}_u(\xi_1,...,\xi_n,q,d,n,m)}{L} \in [0,1], \quad (36)$$

where $L \overset{\text{def}}{=} l_1+l_2+\cdots+l_n$. To find the minimal overall data-transmission intermittency of D/H-SDMC, we start with the minimal data-transmission intermittency for Data-stream u. According to Eq. (35), we need to find the optimal selections of $\mathcal{D}$ and $\mathcal{H}$ (which are represented by $\mathcal{D}^{\text{opt}}$ and $\mathcal{H}^{\text{opt}}$, respectively) and the optimal scalar of λ (which is denoted by λ^{opt}). Since $\mathcal{D}^{\text{opt}} \cup \mathcal{H}^{\text{opt}} = \{2,3,\cdots,n\}$ and $\mathcal{D}^{\text{opt}} \cap \mathcal{H}^{\text{opt}} = \emptyset$, we first derive $\mathcal{D}^{\text{opt}}$, and then determine $\mathcal{H}^{\text{opt}}$ thereupon. However, since the derivation details for $\mathcal{D}^{\text{opt}}$ and λ^{opt} are tedious, we only highlight the crucial results as given by Eqs. (37), (38), (39), and (40) below. Please refer to the appendix in the supplementary document for details.

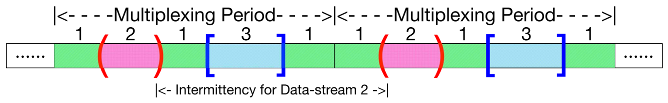

|<- - - -Multiplexing Period- - - ->|<- - - -Multiplexing Period- - - ->|
1 2 1 3 1 1 2 1 3 1
......
|<- Intermittency for Data-stream 2 ->|

Fig. 6. Illustration of data-transmission intermittency in D-SDMC.

Given

$$\mathcal{M}=\begin{cases}\{2,3,\cdots,n\}, & \text{if } m=1,\\ \left\{u\middle|\underset{u=2,3,\ldots,\varphi_1}{\arg\min}\ l'_u\right\}=\left\{u\middle|\underset{u=2,3,\ldots,n}{\arg\min}\ l_u\right\}, & \text{otherwise.}\end{cases} \tag{37}$$

We can find

$$\mathcal{D}^{\text{opt}}=\underset{\mathcal{D}}{\arg\min}\sum_{u\in\mathcal{D}}l_u, \tag{38}$$

and

$$\lambda^{\text{opt}}=\frac{1}{m+1}. \tag{39}$$

According to Eqs. (35), (37), (38), and (39), given $u'=\underset{u=2,3,\ldots,n}{\arg\min}\ l_u$ and $\mathcal{H}^{\text{opt}}\overset{\text{def}}{=}\{u|u\notin\mathcal{D}^{\text{opt}}\cup\{1\}\}$, the minimal data-transmission intermittency for Data-stream u is given by

$$\begin{aligned}&\bar\rho^*_u(\xi_1,\xi_2,\ldots,\xi_n,q,d,n,m)\\&=\begin{cases}\dfrac{\sum_{w=1}^{n}(\mathcal{L}_w+\mathcal{E}_{w,m})-\mathcal{L}_u}{\mathcal{N}(m)}, & u\in\mathcal{D}^{\text{opt}},\\ \dfrac{\sum_{w=1}^{n}(\mathcal{L}_w+\mathcal{E}_{w,m})-\mathcal{L}_u}{2\mathcal{N}(m)}, & u\in\mathcal{H}^{\text{opt}},\\ \dfrac{\sum_{w=1}^{n}(\mathcal{L}_w+\mathcal{E}_{w,m})+(\mathcal{L}_{u'}+\mathcal{E}_{u',m})-\mathcal{L}_1}{(m+1)\mathcal{N}(m)}, & u=1.\end{cases}\end{aligned} \tag{40}$$

According to Eq. (40) for D/H-SDMC, we can further derive the minimal data-transmission intermittencies of all data-streams for D-SDMC and H-SDMC.

For D-SDMC, as shown in Figure 6, the number of pre-delimiters is $n-1$, $\mathcal{D}=\mathcal{D}^{\text{opt}}=\{2,3,\cdots,n\}$, and $\mathcal{H}=\mathcal{H}^{\text{opt}}=\emptyset$. Hence the corresponding grouping matrix $\mathfrak{G}$ as defined in the appendix is an $(n-1)\times(n-1)$ identity matrix here. According to Eq. (39), $\lambda^{\text{opt}}=1/n$. Therefore, given $u'=\underset{u=2,3,\ldots,n}{\arg\min}\ l_u$, the minimal data-transmission intermittency for Data-stream u in D-SDMC is given by

$$\begin{aligned}&\bar\rho^*_u(l_1/n,l_2,\ldots,l_n,q,d,n,n-1)\\&=\begin{cases}\dfrac{\sum_{w=1}^{n}(\mathcal{L}_w+\mathcal{E}_{w,n-1})-\mathcal{L}_u}{\mathcal{N}(n-1)}, & 2\le u\le n,\\ \dfrac{\sum_{w=1}^{n}(\mathcal{L}_w+\mathcal{E}_{w,n-1})+(\mathcal{L}_{u'}+\mathcal{E}_{u',n-1})-\mathcal{L}_1}{n\mathcal{N}(n-1)}, & u=1.\end{cases}\end{aligned} \tag{41}$$

For H-SDMC, as shown in Figure 7, given $u'=\underset{u=2,3,\ldots,n}{\arg\min}\ l_u$, the number of pre-delimiters is 1,

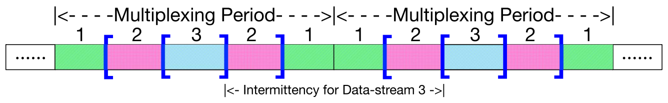

|<- - - -Multiplexing Period- - - ->|<- - - -Multiplexing Period- - - ->|
1 2 3 2 1 1 2 3 2 1
......
|<- Intermittency for Data-stream 3 ->|

Fig. 7. Illustration of data-transmission intermittency in H-SDMC.

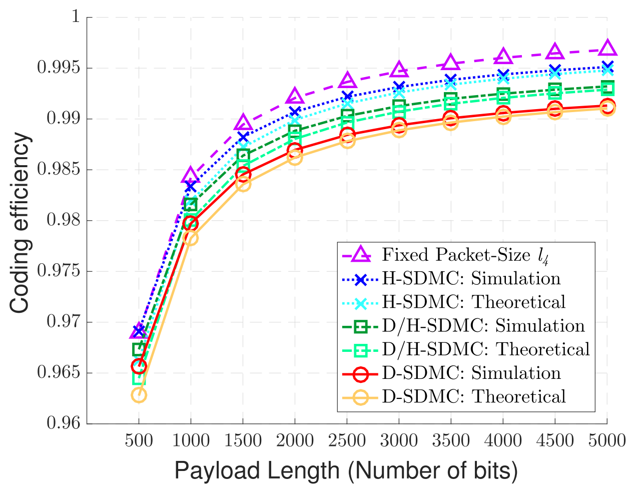


Fig. 8. Coding efficiency comparison of three SDMC schemes when the fixed packet-size l_n is used.

$\mathcal{D}=\mathcal{D}^{\text{opt}}=\{u'\}$, and $\mathcal{H}=\mathcal{H}^{\text{opt}}=\{u\neq u'|2\le u\le n\}$. Hence the corresponding associating matrix as defined in the appendix is $\mathfrak{A}=1$ here (the dimension is 1 in this case). According to Eq. (39), $\lambda^{\text{opt}}=1/2$. Therefore, the minimal data-transmission intermittency for Data-stream u in H-SDMC is given by

$$\begin{aligned}&\bar\rho^*_u(\xi_1,\xi_2,\ldots,\xi_n,q,d,n,1)\\&=\begin{cases}\dfrac{\sum_{w=1}^{n}(\mathcal{L}_w+\mathcal{E}_{w,1})-\mathcal{L}_u}{2\mathcal{N}(1)}, & u\notin\{1,u'\},\\ \dfrac{\sum_{w=1}^{n}(\mathcal{L}_w+\mathcal{E}_{w,1})-\mathcal{L}_u}{\mathcal{N}(1)}, & u\in\{1,u'\}.\end{cases}\end{aligned} \tag{42}$$

According to Eq. (36) and Eq. (40), we can derive the *minimal overall data-tranmission intermittency* for D/H-SDMC as

$$\begin{aligned}&\Psi^*(\xi_1,\ldots,\xi_n,q,d,n,m)\\&\overset{\text{def}}{=}\sum_{u=1}^{n}\frac{l_u\,\bar\rho^*_u(\xi_1,\ldots,\xi_n,q,d,n,m)}{L}\\&=\frac{l_1\left(\sum_{w=2}^{n}\mathcal{L}_w+\mathcal{L}_{u'}\right)}{(m+1)L\mathcal{N}(m)}+\frac{\sum_{u\in\mathcal{D}^{\text{opt}}}l_u\left(\sum_{w=1}^{n}\mathcal{L}_w-\mathcal{L}_u\right)}{L\mathcal{N}(m)}\\&\quad+\frac{\sum_{u\in\mathcal{H}^{\text{opt}}}l_u\left(\sum_{w=1}^{n}\mathcal{L}_w-\mathcal{L}_u\right)}{2L\mathcal{N}(m)}+\frac{l_1\left(\sum_{w=1}^{n}\mathcal{E}_{w,m}+\mathcal{E}_{u',m}\right)}{(m+1)L\mathcal{N}(m)}\\&\quad+\frac{\sum_{u\in\mathcal{D}^{\text{opt}}}l_u\sum_{w=1}^{n}\mathcal{E}_{w,m}}{L\mathcal{N}(m)}+\frac{\sum_{u\in\mathcal{H}^{\text{opt}}}l_u\sum_{w=1}^{n}\mathcal{E}_{w,m}}{2L\mathcal{N}(m)},\end{aligned} \tag{43}$$

where $L=l_1+l_2+\cdots+l_n$.

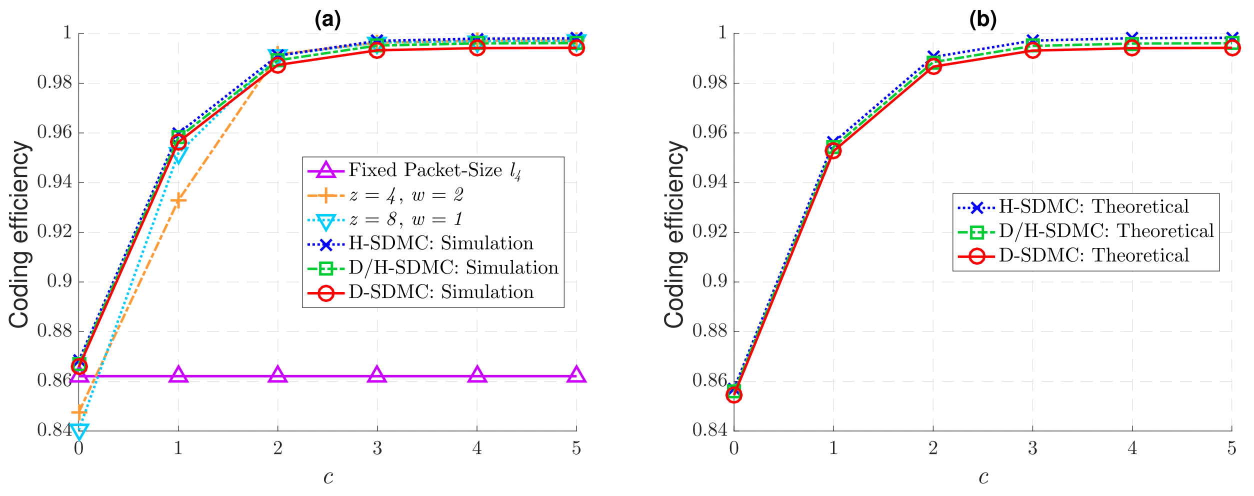


Fig. 9. Coding efficiency comparison of three SDMC schemes when two variable packet-size scenarios are adopted ($z = 4$ and $w = 2$; $z = 8$ and $w = 1$). (a) Simulation results. (b) Theoretical results.

VIII. Numerical Evaluation

In this seciton, all pre-delimiters belong to $\acute{\mathcal{P}}^{q}_{d_1:d-d_1}$ (see Definition 9). The simulation is carried out to benchmark the coding efficiencies and the data-transmission intermittencies for three different SDMC approaches and the existing packet-based method. Assume that a packet-header consists of h bits. Since the packet-body size is fixed, it can be determined by the smallest payload-length $l_{\min} \overset{\text{def}}{=} \min\ \{l_1, l_2, \cdots, l_n\}$. Thus, the total packet-size is $h + l_{\min}$, where $l_u = \beta_u\, l_{\min}$ and $\beta_u \in \mathcal{Z}^+$, $u = 1, 2, \ldots, n$. The coding efficiency of this packet-based method is given by

$$\eta_{\text{packet}}(l_1, ..., l_n : n, h) \overset{\text{def}}{=} \frac{\sum\limits_{u=1}^{n} l_u}{h \sum\limits_{u=1}^{n} \beta_u + \sum\limits_{u=1}^{n} l_u}. \tag{44}$$

In our numerical evaluation, we have four data-streams ($n = 4$). The modulation order is $q = 2$ (binary) and the pre-delimiter length is $d = 10$ bits. The packet-header length is $h = 16$ bits.

We will present the results from our theoretical derivation and Monte Carlo simulation. In the simulation, for each payload length and each data-stream, a thousand random binary payload sequences are generated to benchmark the coding efficiency and the data-transmission intermittency. The theoretical coding efficiencies are given by $\eta(l_1, l_2, l_3, l_4 : 2, 10, 4, 3)$, $\eta(l_1, l_2, l_3, l_4 : 2, 10, 4, 1)$, and $\eta(l_1, l_2, l_3, l_4 : 2, 10, 4, 2)$ for D-SDMC, H-SDMC, and D/H-SDMC, respectively, according to Eq. (21). The theoretical data-transmission intermittencies are given by $\Psi^*(l_1/4, l_2, l_3, l_4, 2, 10, 4, 3)$, $\Psi^*(l_1/2, l_2/2, l_3/2, l_4, 2, 10, 4, 1)$, and $\Psi^*(l_1/3, l_2/2, l_3, l_4, 2, 10, 4, 2)$ for D-SDMC, H-SDMC, and D/H-SDMC, respectively, according to Eq. (43).

Figures 8 and 9 delineate the coding efficiencies in comparison for the fixed packet-size and the variable packet-sizes, respectively. In Figure 8, the payload lengths are $500 \le l_1 = l_2 = l_3 = l_4 \le 5000$. In Figure 9, the payload lengths comply with $l_u = 2^c\, l_{u+1}$, $u = 1, 2, 3$, $l_4 = 100$, and $c = 0, 1, \ldots, 5$. Therefore the coding efficiency of such a packet-based scheme is given by

$$\begin{aligned}\eta_{\text{packet}}(l_1, l_2, l_3, l_4 : 4, h) &= \frac{\sum\limits_{u=1}^{4} l_u}{h \sum\limits_{u=1}^{4} \frac{l_u}{l_4} + \sum\limits_{u=1}^{4} l_u} \\ &= \frac{l_4\, 2^c (2^4 - 1)}{h\, 2^c (2^4 - 1) + l_4\, 2^c (2^4 - 1)} \\ &= \frac{l_4}{h + l_4}.\end{aligned} \tag{45}$$

According to Eq. (45), the coding efficiency of the packet-based scheme is a constant. To further increase this coding efficiency, packets of variable-lengths can be utilized. We assume that z different kinds of packets are utilized. Given an arbitrary $w \in \mathcal{Z}^+$, the packet-sizes can be

$$h + b + l_n\, 2^{wt}, \tag{46}$$

where b is the additional packet-field length to specify which kind of packet is used and $t = 0, 1, \ldots, z-1$. Denote the total number of t^{th} kind of packets necessary for Data-stream u by α_{ut}. Then, the total number of packets for all data-streams is $\sum\limits_{t=0}^{z-1} \alpha_{ut}$. The maximum coding efficiency will be obtained when the following is carried out:

$$\min \sum_{t=0}^{z-1} \alpha_{ut} \quad \text{subject to} \quad l_u = \sum_{t=0}^{z-1} \alpha_{ut}\, l_n\, 2^{wt}, \ \forall u. \tag{47}$$

From Eqs. (44) and (47), the coding efficiency for z variable packet-sizes is given by

$$\eta_{\text{packet}}(l_1, ..., l_n : n, h + b) = \frac{\sum\limits_{u=1}^{n} l_u}{(h + b) \sum\limits_{u=1}^{n} \sum\limits_{t=0}^{z-1} \alpha_{ut} + \sum\limits_{u=1}^{n} l_u}. \tag{48}$$

In Figure 9, we set (*i*) $z = 4$ packet-sizes with $w = 2$ and (*ii*) $z = 8$ packet-sizes with $w = 1$ to compare the coding efficiencies for D-SDMC, H-SDMC, and D/H-SDMC.

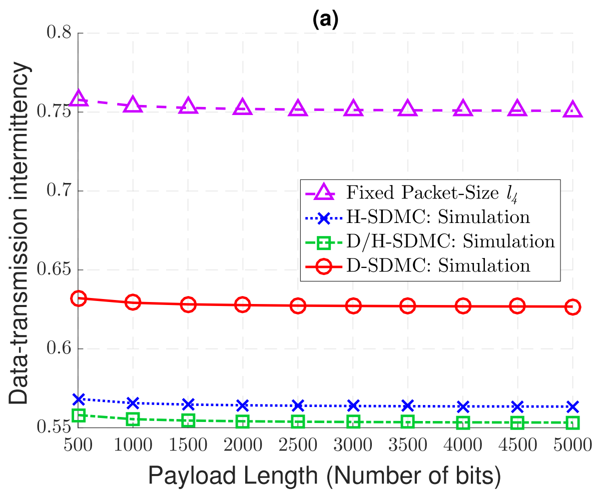


Fig. 10. Overall data-transmission intermittencies of three SDMC schemes when the fixed packet-size l_n is used. (a) Simulation results. (b) Theoretical results.

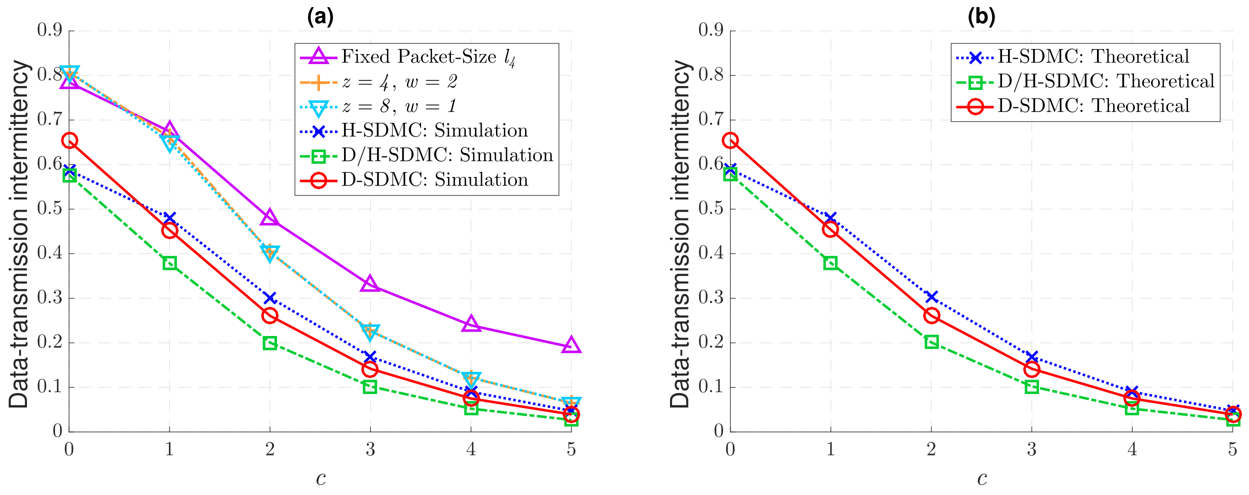


Fig. 11. Overall data-transmission intermittencies of three SDMC schemes when two variable packet-size scenarios are adopted ($z = 4$ and $w = 2$; $z = 8$ and $w = 1$). (a) Simulation results. (b) Theoretical results.

Note that the theoretical and simulation results are very close to each other for every scheme.

Next, according to Eq. (47), we derive the data-transmission intermittency for Data-stream u when z different packet-sizes are used. It is

$$\zeta_u(l_1, ..., l_n, n, h+b) \overset{\text{def}}{=} \frac{(h+b)\sum_{w=1}^{n}\sum_{t=0}^{z-1}\alpha_{wt} + \sum_{w=1}^{n} l_w - l_u}{(h+b)\sum_{w=1}^{n}\sum_{t=0}^{z-1}\alpha_{wt} + \sum_{w=1}^{n} l_w}. \tag{49}$$

The data-transmission intermittency for a single packet-size can be obtained by plugging $z = 1$ and $b = 0$ into Eq. (49). Consequently, the overall data-transmission intermittency for z different packet-sizes is given by

$$\Psi_{\text{packet}}(l_1, ..., l_n, n, h+b) \overset{\text{def}}{=} \sum_{u=1}^{n} \frac{l_u\, \zeta_u(l_1, ..., l_n, n, h+b)}{L}, \tag{50}$$

where $L = l_1 + l_2 + \cdots + l_n$. According to Eq. (50), we depict the overall data-transmission intermittencies of the three SDMC schemes in comparison for the fixed packet-size and the variable packet-sizes in Figures 10 and 11, respectively. According to Figures 8-11, the theoretical and simulation results for both coding efficiency and overall intermittency are very close to each other for each SDMC scheme. Therefore, they validate our derived formulae. It is desirable to have a large coding efficiency and a small intermittency. However, we can observe from Figures 8-11 that there exists a trade-off between these two metrics. That is, the H-SDMC scheme facilitates the largest coding efficiency but the D/H-SDMC scheme leads to the smallest overall intermittency instead.

In the appendix, we provide two more examples to illustrate in detail how to apply our new approach to design delimiters and how to utilize our theoretical analysis to evaluate the three SDMC schemes. These two examples highlight the crucial results from this paper.

IX. Conclusion

In this paper, we establish a complete framework and analysis for software-defined multiplexing code (SDMC), which can play a crucial part in the future software-defined networks. The SDMC approach is free of any packet format and very flexible for any kind of network protocol. The key elements in the SDMC, namely pre-delimiters, can be constructed from fast-searchable strings (FSSs) or semi-fast-searchable strings (SFSSs). Once either kind of strings are used as pre-delimiters, the computational complexity for stuffing the original payload is reduced significantly, from quadratic- to linear-time. Meanwhile, we propose three different SDMC schemes, namely distributed (D-SDMC), hierarchical (H-SDMC), and hybrid (D/H-SDMC) schemes. Two important performance measures, coding efficiency and data-transmission intermittency, are proposed to benchmark these three SDMC schemes and the conventional packet-based method. The theoretical formulae of these two performance measures are derived and compared with the simulation results. The three SDMC schemes outperform the conventional packet-based method in terms of coding efficiency and data-transmission intermittency. According to our numerical evaluation, there exists a trade-off between these two metrics. That is, the H-SDMC scheme can give rise to the largest coding efficiency but the D/H-SDMC scheme is inflicted with the smallest overall intermittency instead.

References

[1] C. Metter, M. Seufert, F. Wamser, T. Zinner, and P. Tran-Gia, "Analytical model for SDN signaling traffic and flow table occupancy and its application for various types of traffic," *IEEE Trans. Netw. Service Manage.*, vol. 14, no. 3, pp. 603–615, Sep. 2017.

[2] G. Levy, S. Pontarelli, and P. Reviriego, "Flexible packet matching with single double cuckoo hash," *IEEE Commun. Mag.*, vol. 55, no. 6, pp. 212–217, Jun. 2017.

[3] T.-S. Chen, D.-Y. Lee, T.-T. Liu, and A.-Y. Wu, "Dynamic reconfigurable ternary content addressable memory for OpenFlow-compliant low-power packet processing," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 63, no. 10, pp. 1661–1672, Oct. 2016.

[4] N. Kitsuwan, S. Ba, E. Oki, T. Kurimoto, and S. Urushidani, "Flows reduction scheme using two MPLS tags in software-defined network," *IEEE Access*, vol. 5, pp. 14626–14637, 2017.

[5] D. Hu, S. Li, H. Huang, W. Fang, and Z. Zhu, "Flexible flow converging: A systematic case study on forwarding plane programmability of protocol-oblivious forwarding (POF)," *IEEE Access*, vol. 4, pp. 4707–4719, 2016.

[6] B. Leng, L. Huang, X. Wang, H. Xu, and Y. Zhang, "A mechanism for reducing flow tables in software defined network," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2015, pp. 5302–5307.

[7] X. Jia, Y. Jiang, Z. Guo, and Z. Wu, "Reducing and balancing flow table entries in software-defined networks," in *Proc. IEEE 41st Conf. Local Comput. Netw. (LCN)*, Nov. 2016, pp. 575–578.

[8] T. Wang, F. Liu, and H. Xu, "An Efficient online algorithm for dynamic SDN controller assignment in data center networks," *IEEE/ACM Trans. Netw.*, vol. 25, no. 5, pp. 2788–2801, Oct. 2017.

[9] S. A. Astaneh and S. S. Heydari, "Optimization of SDN flow operations in multi-failure restoration scenarios," *IEEE Trans. Netw. Service Manage.*, vol. 13, no. 3, pp. 421–432, Sep. 2016.

[10] H. Xu, Z. Yu, C. Qian, X.-Y. Li, and L. Huang, "Minimizing flow statistics collection cost using wildcard-based requests in SDNs," *IEEE/ACM Trans. Netw.*, vol. 25, no. 6, pp. 3587–3601, Dec. 2017.

[11] T. K. Phan, D. Griffin, E. Maini, and M. Rio, "Utility-centric networking: Balancing transit costs with quality of experience," *IEEE/ACM Trans. Netw.*, vol. 26, no. 1, pp. 245–258, Feb. 2018.

[12] E. Sakic and W. Kellerer, "Response time and availability study of RAFT consensus in distributed SDN control plane," *IEEE Trans. Netw. Service Manage.*, vol. 15, no. 1, pp. 304–318, Mar. 2018.

[13] P. Wang *et al.*, "Minimizing controller response time through flow redirecting in SDNs," *IEEE/ACM Trans. Netw.*, vol. 26, no. 1, pp. 562–575, Feb. 2018.

[14] P. Wang, H. Xu, L. Huang, J. He, and Z. Meng, "Control link load balancing and low delay route deployment for software defined networks," *IEEE J. Sel. Areas Commun.*, vol. 35, no. 11, pp. 2446–2456, Nov. 2017.

[15] H. Xu *et al.*, "Joint route selection and update scheduling for low-latency update in SDNs," *IEEE/ACM Trans. Netw.*, vol. 25, no. 5, pp. 3073–3087, Oct. 2017.

[16] H. Zhou, C. Wu, Q. Cheng, and Q. Liu, "SDN-LIRU: A lossless and seamless method for SDN inter-domain route updates," *IEEE/ACM Trans. Netw.*, vol. 25, no. 4, pp. 2473–2483, Aug. 2017.

[17] S. Vissicchio, L. Vanbever, L. Cittadini, G. G. Xie, and O. Bonaventure, "Safe update of hybrid SDN networks," *IEEE/ACM Trans. Netw.*, vol. 25, no. 3, pp. 1649–1662, Jun. 2017.

[18] M. Azizian, S. Cherkaoui, and A. S. Hafid, "Vehicle software updates distribution with SDN and cloud computing," *IEEE Commun. Mag.*, vol. 55, no. 8, pp. 74–79, Aug. 2017.

[19] S. Vissicchio and L. Cittadini, "Safe, efficient, and robust SDN updates by combining rule replacements and additions," *IEEE/ACM Trans. Netw.*, vol. 25, no. 5, pp. 3102–3115, Oct. 2017.

[20] T. Dargahi, A. Caponi, M. Ambrosin, G. Bianchi, and M. Conti, "A survey on the security of stateful SDN data planes," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 3, pp. 1701–1725, Mar. 2017.

[21] M. Ambrosin, M. Conti, F. De Gaspari, and R. Poovendran, "LineSwitch: Tackling control plane saturation attacks in software-defined networking," *IEEE/ACM Trans. Netw.*, vol. 25, no. 2, pp. 1206–1219, Apr. 2017.

[22] J. H. Cox, R. Clark, and H. Owen, "Leveraging SDN and WebRTC for rogue access point security," *IEEE Trans. Netw. Service Manage.*, vol. 14, no. 3, pp. 756–770, Sep. 2017.

[23] C. Qi, J. Wu, G. Cheng, J. Ai, and S. Zhao, "An aware-scheduling security architecture with priority-equal multi-controller for SDN," *China Commun.*, vol. 14, no. 9, pp. 144–154, Sep. 2017.

[24] C.-L. Hsieh and N. Weng, "Many-field packet classification for software-defined networking switches," in *Proc. ACM/IEEE Symp. Archit. Netw. Commun. Syst. (ANCS)*, Mar. 2016, pp. 13–24.

[25] K. G. Pérez, X. Yang, S. Scott-Hayward, and S. Sezer, "A configurable packet classification architecture for software-defined networking," in *Proc. 27th IEEE Int. System-on-Chip Conf. (SOCC)*, Sep. 2014, pp. 353–358.

[26] K. G. Pérez, X. Yang, S. Scott-Hayward, and S. Sezer, "Optimized packet classification for software-defined networking," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2014, pp. 859–864.

[27] E. Alasadi and H. S. Al-Raweshidy, "SSED: Servers under software-defined network architectures to eliminate discovery messages," *IEEE/ACM Trans. Netw.*, vol. 26, no. 1, pp. 104–117, Feb. 2018.

[28] J. Saldana *et al.*, "Small-packet flows in software defined networks: Traffic profile optimization," *J. Netw.*, vol. 10, no. 4, pp. 176–187, Apr. 2015.

[29] S. C. H. Huang and H. C. Wu, "Software-defined multiplexing codes," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2015, pp. 1–6.

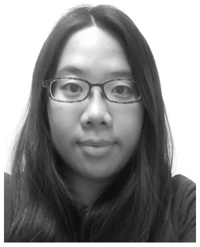
[30] F. Checconi, L. Rizzo, and P. Valente, "QFQ: Efficient packet scheduling with tight guarantees," *IEEE/ACM Trans. Netw.*, vol. 21, no. 3, pp. 802–816, Jun. 2013.

[31] L. Huang and M. J. Neely, "Utility optimal scheduling in energy-harvesting networks," *IEEE/ACM Trans. Netw.*, vol. 21, no. 4, pp. 1117–1130, Aug. 2013.

[32] Z. Mao, C. E. Koksal, and N. B. Shroff, "Optimal Online scheduling with arbitrary hard deadlines in multihop communication networks," *IEEE/ACM Trans. Netw.*, vol. 24, no. 1, pp. 177–189, Feb. 2016.

[33] G. C. Sankaran and K. M. Sivalingam, "Design and analysis of scheduling algorithms for optically groomed data center networks," *IEEE/ACM Trans. Netw.*, vol. 25, no. 6, pp. 3282–3293, Dec. 2017.

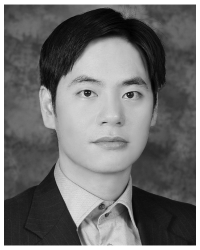
[34] D. E. Knuth, J. H. Morris, Jr., and V. R. Pratt, "Fast pattern matching in strings," *SIAM J. Comput.*, vol. 6, no. 2, pp. 323–350, Jul. 1977, doi: 10.1137/0206024.



Elaine Y.-N. Sun received the B.S. degree in mathematics from National Tsing Hua University, Taiwan, in 2014, where she is currently pursuing the Ph.D. degree in communication engineering. Her research interests include error-correcting codes, blockchain, wireless networking, cryptography, wireless protocols, and quantum computers.

Hsiao-Chun Wu (M'00–SM'05–F'15) received the B.S.E.E. degree from National Cheng Kung University, Taiwan, in 1990, and the M.S. and Ph.D. degrees in electrical and computer engineering from the University of Florida, Gainesville, in 1993 and 1999, respectively.

From March 1999 to January 2001, he was a Senior Electrical Engineer with Motorola Personal Communications Sector Research Labs. Since January 2001, he has been a Faculty Member with the Department of Electrical and Computer Engineering, Louisiana State University, Baton Rouge, LA, USA. From July to August 2007, he was a Visiting Assistant Professor with the Television and Networks Transmission Group, Communications Research Centre, Ottawa, ON, Canada. From August to December 2008, he was a Visiting Associate Professor with the Department of Electrical Engineering, Stanford University, CA, USA. He has published more than 240 peer-refereed technical journal and conference papers in electrical and computer engineering. His research interests include wireless communications and signal processing. He is an IEEE Distinguished Lecturer. He has also served for numerous textbooks, IEEE/ACM conferences, and journals as a technical committee member, the symposium chair, the track chair, or a reviewer of signal processing, communications, and circuits and computers. He has served as an Associate Editor for the IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY, the IEEE COMMUNICATIONS LETTERS, and the IEEE SIGNAL PROCESSING LETTERS. He currently serves as an Editor for the IEEE TRANSACTIONS ON WIRELESS COMMUNICATIONS, an Associate Editor for the IEEE TRANSACTIONS ON BROADCASTING, the IEEE TRANSACTIONS ON MOBILE COMPUTING, and the IEEE TRANSACTIONS ON SIGNAL PROCESSING, and a Technical Editor for the *IEEE Communications Magazine*.



Scott C.-H. Huang received the B.S. degree in mathematics from National Taiwan University in 1998, and the Ph.D. degree in computer science engineering from the University of Minnesota, Twin Cities, in 2004. He is currently an Associate Professor with the Department of Electrical Engineering, Institute of Communication Engineering, National Tsing Hua University (NTHU), Taiwan. Prior to joining NTHU, he was with the City University of Hong Kong from 2005 to 2010. His research interests include FinTech/blockchain, information security, wireless networking, digital signal processing, and error-correcting codes.