A Storage Algorithm of Code Parameters in Embedded System Based on Dynamic Programming

Xu Na, Zhang Xiaotong, Zhang Yan, Yuan Lingling, Zhang Lei
School of Information Engineering
University of Science and Technology Beijing
Beijing, China
xunause@sohu.com

Hu Guolin
Beijing Aerospace Control Center
Beijing, China

***Abstract*—In embedded system, how to utilize the limited physical storage capacity for code parameters is a challenging problem. This paper proposes a dynamic programming based algorithm to solve this problem under DOCSIS standard. The algorithm is able to generate the best storage structure of parameters in terms of the width of the bus, the priority and least storage spaces of those parameters. The algorithm has been real-implemented in the network devices, and the evaluations show that it can significantly improve the utilization of the physical storage and reduce hardware cost.**

Keywords-Embedded System; DOCSIS; Dynamic Programming; Storage Algorithm

I. Introduction

HFC (Hybrid Fiber-Coax) is widely used broadband access technology. The greatest wire television network of the world locates in China. As the development of the CATV (community antenna television), network chip which includes two-way transmission function and commutative set-top box become the key technology and device [1].

The transmission on MAC layer of DOCSIS (Data-Over-Cable Service Interface Specifications) [2] is an important technology that must been implemented. The packets of MAC layer include a great many parameters encoded in a type/length/value (TLV) form. These parameters analyzed by software are stored in physical memories, for embedded system to use. An optimal strategy of storage is proposes to improve the efficiency of the utilization of the storage capacity, reduce the hardware cost, and hence increase the speed of access. DOCSIS standard only specifies the code format. However, it doesn't give out how to store those parameters in embedded system [3].

The rest of the paper is organized as follow: Section 2 introduces the topology of HFC network. Section 3 introduces the optimal strategy of storage. Section 4 introduces auto-generation storage structure algorithm. Finally, the performance analysis and test results are introduced.

II. Topology and Transmission of HFC Network

HFC network is a tree branch structure. There are three parts in that: HFC transmission media, several CM (cable modem) and a CMTS (cable modem termination system). The CMTS is the central management device. The most distinct characteristic of HFC is the independence between upstream channels and downstream ones.

A. *The topology of HFC network*

In Figure 1, on the downstream channel, the CMTS is the only sender which sends down broadcast packets sequentially [4]. Each of the CMs is allocated a special address, and only receives the packets with its own address. On upstream channel, the packets from those different CMs to the CMTS are sent by the same media. Any upstream channels are all shared by several CMs, so a management is needed to control the transmission in order.

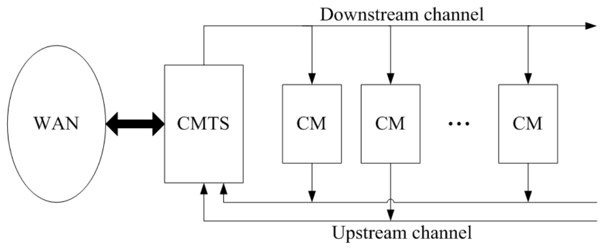


Figure 1. Topology of HFC network

B. *The transmission of upstream*

The CMTS defines the mini-slots to be able to control the transmission on the upstream channel [5]. A transmit opportunity is defined as any mini-slot in which a CM may be allowed to start a transmission. Transmit opportunities typically apply to contention opportunities and are used to calculate the proper amount to defer in the contention resolution process. The CMTS controls assignments on the upstream channel through the MAP of bandwidth allocation and determines which mini-slots are subject to collisions. The CMTS may allow collisions on either requests or data[6].

C. *The code of the parameters*

An Upstream Channel Descriptor (UCD) must be transmitted by the CMTS at a periodic interval to define the characteristics of an upstream channel (see Figure 2). A separate message must be transmitted for each active upstream. To provide for flexibility the message parameters following the channel ID must be encoded in a type/length/value (TLV) form in which the type and length fields are each 1 octet long.

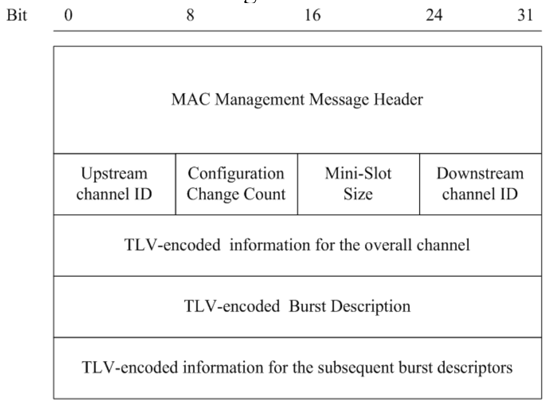

Bit
0
8
16
24
31
MAC Management Message Header
Upstream channel ID
Configuration Change Count
Mini-Slot Size
Downstream channel ID
TLV-encoded information for the overall channel
TLV-encoded Burst Description
TLV-encoded information for the subsequent burst descriptors

Figure 2. Upstream Channel Descriptor

A burst descriptor must be included for each interval usage code that is to be used in the allocation MAP of bandwidth allocation. Within each burst descriptor is an unordered list of physical-layer attributes, encoded as TLV values. These attributes are shown in Table 1. The CMTS must ensure that the set of burst attributes for all the burst descriptors in the UCD allow any CM on the upstream to be able to request enough mini-slots to be able to transmit a maximum size packet. According to the valid scale of each value, the minimum size to store parameter can be calculated out as its hardware bit-space. Those attributes with the value of 1 or 2, could be represented by 1 or 0 respectively, only costing 1-bit space.

TABLE I. UPSTREAM PHYSICAL-LAYER BURST ATTRIBUTE

Space (bit)	**TLV parameters**		
	Type	***Length***	***Value***
3	1	1	1~6
1	2	1	1 or 2
11	3	2	Max(1536)
11	4	2	0~1536
5	5	1	0~16
8	6	1	16~253
16	7	2	0~15
8	8	1	0~255
8	9	1	0~255
1	10	1	1 or 2
1	11	1	1 or 2
8	12	1	8
12	13	2	2×Nr~2048
1	14	1	1 or 2
1	15	1	1 or 2
8	16	1	1~128
5	17	1	1~31
1	18	1	1 or 2

III. OPTIMAL STRATEGY OF STORAGE

To optimize the storage of parameters, there are two targets as follows:

- Making the storage space smaller, can reduce the hardware cost and improve the utilization of the physical storage.
- Making the storage structure reasonable, can ensure parameters accessed much frequently in faster memory, in order to make the average access time shorter.

The size of a physical block which can be accessed in one time is determined by the width of the system bus. If a bus is 32-bit, the block is 4-byte. The access frequency of each parameter maybe as priority from 1 to n, can be calculated by the statistics of attributes [7].

The simplest way to store is to allocate memory as the length fields of the parameters. For example, a burst descriptor is 22 bytes [8][9]. If hoping to get the parameters accessed only in one time without any mask operation, a burst descriptor is allocated 72 bytes, when the width of bus is 32-bit, or 36 bytes of 16-bit bus. Through analyzing the valid scale of each parameter, the smallest storage space of a burst descriptor is only 109 bits, equal to 14 bytes. In actual project, we adopt the structure in Figure 3 due to tighter storage would bring the access to the parameters more complex.

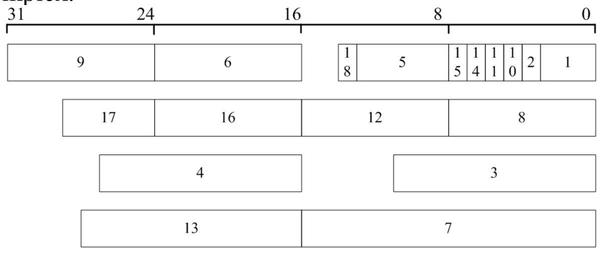


Figure 3. Storage structure

The storage structure in Figure 4 is based on 16-bit bus, but it can be adapted on 32-bit bus. If the system bus is 16-bit, there are 10 parameters which need to moved, when they are accessed. If the system bus is 32-bit, the number of these parameters is 14. If the system bus is 8-bit, there are 5 parameters having to address 2 times and contact the fragment data when they are accessed. However, these operations are inevitable in any situation.

Figure 4 shows the six kinds of storage strategies: a-letter represents no mask operation and storing one parameter from the lowest bit; b-letter represents having mask operation when some parameters are accessed.

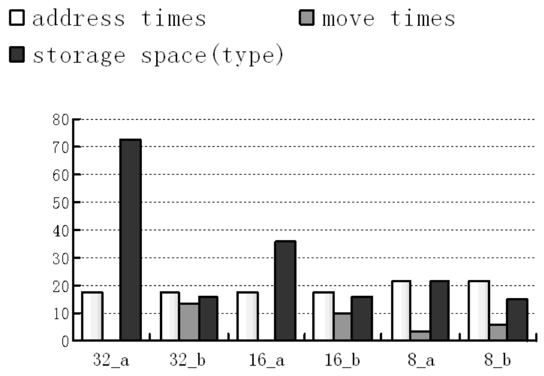


Figure 4. Comparing the storage strategies

By analyzing the architecture of the system and the statistics, these conclusions can be got as follow:

- With adding the mask operation, the total size of the spares can reduce.
- Without the mask operation, as reducing the width of bus, the storage spaces will reduce; with the mask operation, the storage space is not relative with the width of bus.
- As the width of bus reduces, the times of address for access these parameters increase.
- With the mask operation, the move operation is inevitable; without the mask operation, as the width of bus increases, the probability of the move operation reduces.

IV. AUTO-GENERATION STORAGE STRUCTURE ALGORITHM

In mathematics and computer science field, dynamic programming is a method of solving problems exhibiting the properties of overlapping sub-problems and optimal substructure that takes much less time than naive methods. The term was originally used in the 1940s by Richard Bellman to describe the process of solving problems where one needs to find the best decisions one after another. By 1953, he had refined this to the modern meaning [10]. The field was founded as a systems analysis and engineering topic which is recognized by the IEEE. Bellman's contribution is remembered in the name of the Bellman equation, a central result of dynamic programming which restates an optimization problem in recursive form.

A. Design of algorithm

In the process of searching an optimal strategy to store parameters, the problem can be abstracted like the Knapsack Problem. The Knapsack Problem is a problem in combinatorial optimization. It derives its name from the following maximization problem of the best choice of essentials that can fit into one bag to be carried on a trip. Given a set of items, each with a cost and a value, determine the number of each item to include in a collection so that the total cost is less than a given limit and the total value is as large as possible. A similar problem often appears in business, combinatory, complexity theory, cryptography and applied mathematics.

In the following, we have *n* kinds of items, from 1 to *n*. Each item *i* has a value c_i and a weight w_i. The maximum weight that we can carry in the bag is *M*. The 0-1 knapsack is a special case of the original knapsack problem in which each item of input cannot be subdivided to fill a container in which that input partially fits. The 0-1 knapsack problem restricts the number of each kind of item, x_i, to zero or one. Mathematically the 0-1 knapsack problem can be formulated as:

$$\text{Maximize } Z = \sum_{i=1}^{n} c_i \times x_i \qquad (1)$$

$$\text{Subject to } \sum_{i=1}^{n} w_i \times x_i \le M(x_i = 0 \wedge x_i = 1) \qquad (2)$$

Putting the storage problem into the model of the 0-1 knapsack problem, the block which can be accessed at one time is the limited maximum weight of knapsack. Those parameters are like the items and the priority of the parameters is like the value of items.

B. Realization of algorithm

Algorithm 1: Finding out the optimal structure of storage in one block.

```
Function Z(n,M);
    If m<0 then z=Minnum exit
        Else Z1=Z(n-1,m);
    Z2=Z(n-1,m-Wn)+Cn;
    Z=max(Z1,Z2);
End;
```

Algorithm 2: Multi-call for realizing all the parameters' optimal storage.

```
While (n>0)
    Z(n,M);
    Put out the result of Z(n,M);
    j=0;
    For i=1 to n step 1
        If (parameter[i] is not in the block) then
            j=j+1;
            Insert the parameter[i] into queue as parameter[j];
        End if;
    Next i;
    n=j;
End while;
```

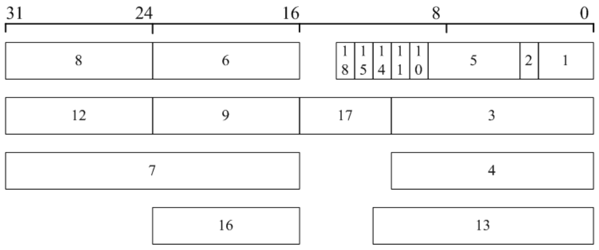
V. TEST AND EVALUATION

From the architecture of the embedded system, we can get the size of one block, according to the width of bus. On the initial stage, the priority is input at random. After a long time system running we can get the statistics of the access frequency, these statistics would determine the priority of parameters. We had tested three kinds of the situations in the terms of different buses, 8-bit, 16-bit and 32-bit. Figure 5 shows the output result of the program of 16-bit bus.

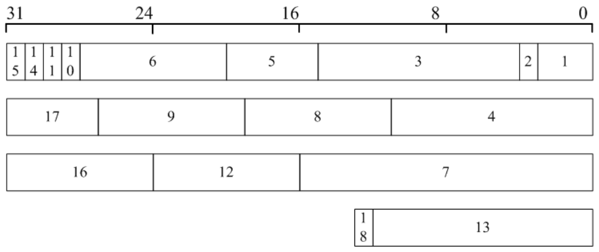
```
n = 18  M = 16
block 1 : 1  2  5  10  11  14  15  18  --> bits = 14
block 2 : 6  8  --> bits = 16
block 3 : 3  17  --> bits = 16
block 4 : 9  12  --> bits = 16
block 5 : 4  --> bits = 11
block 6 : 7  --> bits = 16
block 7 : 13  --> bits = 12
block 8 : 16  --> bits = 8
```

Figure 5. The result of 16-bit bus

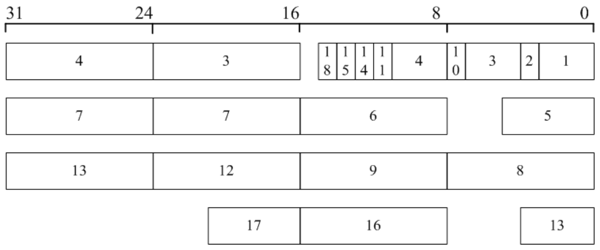
Figure 6 shows the storage structure of the different width of bus.



(a) The storage structure of 16-bit bus



(b) The storage structure of 32-bit bus



(c) The storage structure of 8-bit bus

Figure 6. The storage structure of the different buses

The result data reported here suggest that using the optimal strategy of storage can improve the utilization of the physical storage to 50%. The algorithm is very economical and reliable.

CONCLUSION

This paper presents a storage algorithm of code parameters in embedded system based on dynamic programming. Saving memory resource and reducing hardware cost are the main principles of the embedded system design. This algorithm can automatically generate the optimal strategy of storage for designers. It not only achieves the design targets, but also improves the efficiency of developing embedded system. Through real-implemented in the network devices, this algorithm is proved that is reliable and satisfactory.

REFERENCES

[1] Y. Lin, C. Huang and W. Yin, "Allocation and scheduling algorithms for IEEE 802.14 and MCNS in hybrid fiber coaxial networks," IEEE Transactions on Broadcasting, vol. 44, Apr. 1998, pp. 427–435.

[2] Cable Television Laboratories, Inc.. Data-Over-Cable Service Interface Specifications-Radio Frequency Interface Specification. 2002.

[3] G. Sater and K. Stambaugh, "Media access control protocol based on DOCSIS 1.1," IEEE 802.16 Broadband Wireless Access Working Group, 1999.

[4] D. Fellows and D. Jones, "DOCSIS cable modem technology," IEEE Communications Magazine, vol. 39, Mar. 2001, pp. 202–209.

[5] R. Domdom, B. Espey, M. Goodman, K. Jones, V. Lim and S. Patek, "Transient analysis of DOCSIS 1.1 cable modem networks," In: Proceedings of 2000 IEEE International Conference on Systems, Man, and Cybernetics, Mashville, TN, USA, Mar. 2000, pp. 2263–2268.

[6] M. D. Carroll, "Aligning the initial maintenance intervals of cable modem upstream channels," IEEE Communications Magazine, vol. 41, Sep. 2003, pp. 140–146.

[7] W. Kuo, S. Kumar and C. Kuo, "Improved priority access, bandwidth allocation and traffic scheduling for DOCSIS cable networks," IEEE Transactions on Broadcasting, vol. 49, Apr. 2003, pp. 371–382.

[8] W. Liao and H. Ju, "Adaptive slot allocation in DOCSIS-based CATV networks," IEEE Transactions on Multimedia, vol. 6, Mar. 2004, pp. 479–488.

[9] M. Siahaan and A. H. Gunawan, "Determination Consideration for DOCSIS Implementation," In: Proceedings of 2002 Asia-Pacific Conference on Circuits and Systems(APCCAS'02), Singapore, Jan. 2002, pp. 431–434.

[10] J. Adda and R. Cooper, 2003. Dynamic Economics. MIT Press. An accessible introduction to dynamic programming in economics. The link contains sample programs.