·体系结构与软件技术· 文章编号：1000-3428(2020)10-0210-06 文献标志码：A 中图分类号：TP311

一种改进的固件增量更新算法

王豫新[a,b]，高美凤[a,b]

（江南大学 a. 轻工过程先进控制教育部重点实验室；b. 物联网工程学院，江苏 无锡 214122）

摘 要： 针对 bsdiff 算法在嵌入式设备固件更新中构建新版本固件时内存消耗大的问题，提出一种节约内存的增量更新算法。利用改进的 bsdiff 算法的补丁文件格式，避免应用补丁文件时记录并频繁计算地址偏移量。将 bsdiff 算法中的并行解压过程更换为串行解压，并通过分批处理数据以减小需要的辅助空间。同时，将非对称的无损压缩算法应用到改进后的增量更新算法的压缩与解压缩过程，降低由于解压缩补丁文件而造成的较大内存消耗。实验结果表明，与 bsdiff 算法、xdelta 算法、vcdiff 算法、zdelta 算法相比，该算法能够有效减少构建新版本固件时的内存消耗，且具有良好的压缩性能。

关键词： 增量更新；内存消耗；固件；嵌入式设备；无损压缩；补丁文件

开放科学（资源服务）标志码（OSID）：

中文引用格式： 王豫新，高美凤. 一种改进的固件增量更新算法[J]. 计算机工程，2020，46(10)：210-215.

英文引用格式： WANG Yuxin，GAO Meifeng. An improved incremental update algorithm for firmware[J]. Computer Engineering，2020，46(10)：210-215.

An Improved Incremental Update Algorithm for Firmware

WANG Yuxin[a,b]，GAO Meifeng[a,b]

（a. Key Laboratory of Advanced Process Control for Light Industry（Ministry of Education）；
b. School of Internet of Things Engineering，Jiangnan University，Wuxi，Jiangsu 214122，China）

【Abstract】 In order to solve the high memory consumption of bsdiff algorithm when building new versions of firmware in the firmware update of embedded devices，this paper proposes an incremental update algorithm that saves memory. The improved patch file format of the bsdiff algorithm is used to avoid recording and calculating the address offset frequently in the application of the patch files. The parallel decompression process in the bsdiff algorithm is replaced by serial decompression，and the required auxiliary space is reduced by processing data in batches. At the same time，the asymmetric lossless compression algorithm is applied to the compression and decompression process of the improved incremental update algorithm，which reduces the memory consumption caused by the decompression of patch files. Experimental results show that，compared with bsdiff algorithm，xdelta algorithm，vcdif algorithm and zdelta algorithm，the proposed algorithm can effectively reduce the memory consumption when building new versions of firmware，and has good compression performance.

【Key words】 incremental update；memory consumption；firmware；embedded device；lossless compression；patch file

DOI： 10.19678/j.issn.1000-3428.0056426

0 概述

随着物联网技术的不断发展，嵌入式设备的数量大幅增加，2025 年预计将会有 1 万亿台～3 万亿台的嵌入式设备投入使用到物联网场景中[1]。固件是嵌入式设备中基础、底层的工作软件[2]，其安全性与稳定性极其重要，只有对固件提供持续的更新服务，才能确保设备的安全性，满足用户对设备功能的需求。固件更新主要有全量更新和增量更新 2 种更新方式。全量更新方式是将新版本的固件完整发送给设备，不仅会造成传输过程中的流量浪费，还会增加设备的能源消耗[3]，而增量更新方式仅发送新、旧版本之间的差异文件给设备，因此能够以较小的流量消耗和能源消耗，实现远程修复固件的安全漏洞和升级固件功能[4]。

基金项目： 国家自然科学基金（61373126）。

作者简介： 王豫新（1993—），男，硕士研究生，主研方向为嵌入式系统应用与开发、物联网技术；高美凤，副教授、博士。

收稿日期： 2019-10-28 **修回日期：** 2019-12-02 **E-mail：** wangyu_x@163.com

增量更新的基本流程是开发端使用输入的新、旧版本固件生成补丁文件,设备端使用接收到的补丁文件和已有的旧版本固件恢复新版本固件。bsdiff 算法[5]是较为成熟的增量更新算法,由于该算法生成的补丁文件较小,因此而得到广泛应用。如谷歌使用 bsdiff 算法的变体算法更新全球 Google Chrome 浏览器,文献[6]将 bsdiff 算法应用于更新移动互联网个人云存储系统,文献[7]采用 bsdiff 算法实现手机操作系统的更新。但是 bsdiff 算法在待更新设备上执行构建新版本固件任务时的内存消耗较大,且随着新版本固件体积的增大而不断增大,因此并不适用于内存资源受限的嵌入式设备。bsdiff 算法执行构建新版本固件任务需要的辅助空间为 $u+v+w$,其中,u、v、w 分别为旧版本固件大小、新版本固件大小与补丁文件的大小。文献[8]通过重新排序构建新版本固件要执行的编辑操作将其需要的辅助空间减小为 $\max(u,v)+w$。文献[9]通过改进 bsdiff 算法的补丁文件格式,避免了在应用补丁文件时将整个补丁文件读入内存以及频繁计算偏移量。文献[10]利用哈希算法检测出新、旧版本文件之间的差异部分,采用 bsdiff 算法分块执行增量更新,减小了 bsdiff 算法需要的新、旧版本固件大小的辅助空间。同时,由于增量更新算法生成补丁文件的过程发生在运算能力强、内存资源丰富的 PC 端,而应用补丁构建新版本固件的过程发生在计算能力弱、内存资源有限的嵌入式设备端,这 2 个过程具有典型的非对称特性[11],因此将具备非对称特性的无损压缩算法引入 MS-bsdiff 算法的压缩、解压缩过程,使得 MS-bsdiff 算法不仅能生成较小的补丁文件,还能以较小的内存消耗在设备端完成增量更新。

本文基于 bsdiff 算法,提出一种节约内存的 MS-bsdiff(Memory Saving-bsdiff)增量更新算法。该算法通过改进 bsdiff 算法的补丁文件格式,将 bsdiff 算法中的并行解压过程替换为串行解压,并减小申请的辅助空间,以降低设备端构建新版本固件时的内存消耗。

1　bsdiff 增量更新算法

1.1　bsdifff 算法描述

根据 bsdiff 算法的具体应用,可将其分为 2 个子过程:

1)diff 过程:在开发端计算新版本固件与旧版本固件之间的差异,并对该差异进行编码和压缩,生成补丁文件。

2)patch 过程:在设备端解压缩补丁文件中指定的数据,并按照相应的解码规则应用补丁文件和旧版本固件,从而构建出新版本固件。

图 1 为 bsdiff 算法的补丁文件的格式,其中,阴影部分表示被压缩的数据区域。

图 1　bsdiff 算法的补丁文件格式

Fig. 1　Patch file format of bsdiff algorithm

bsdiff 算法的 diff 过程具体步骤为:

步骤 1　创建空的补丁文件,申请 2 个辅助空间 a 和 b。

步骤 2　通过后缀排序以及二分查找算法检索如图 2 所示的 old、new 文件中的相似匹配区域与不匹配区域。

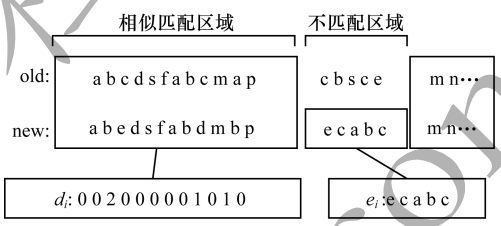

图 2　bsdiff 算法的编码

Fig. 2　Coding of bsdiff algorithm

步骤 3　对相似匹配区域与不匹配区域的数据进行编码。首先,计算 old、new 文件的相似匹配区域的字符差值,并生成一个 d_i 数据块,将 d_i 数据块以追加的方式写入辅助空间 a。其次,不匹配区域代表 new 文件中独有的数据块,将该数据块编码为一个 e_i 数据块,并将 e_i 数据块以追加的方式写入辅助空间 b。最后,将生成的当前 d_i 数据块和 e_i 数据块用于执行 patch 过程的控制信息,并将这些控制信息存放于一个三元数组 $c_i(x_i,y_i,z_i)$ 中,其中,x_i 表示 d_i 的字节数,y_i 表示 e_i 的字节数,z_i 表示 old 文件中的偏移量。

步骤 4　将 c_i 中的数据压缩后直接写入补丁文件的 ctrl 区域。

步骤 5　重复步骤 1 ~ 步骤 4,直至 new 文件的末尾。

步骤 6　分别压缩辅助空间 a 与辅助空间 b 中的数据,依次写入补丁文件的 diff 区域、extra 区域,并补充文件头部信息,得到完整的补丁文件。

bsdiff 算法执行对应的 patch 过程具体步骤为:

步骤 1　创建空的 new 文件,申请需要的辅助空间 a[newsize]、辅助空间 b[oldsize]。

步骤 2　打开 3 个并行的解压进程,并分别用于解压缩补丁文件的 ctrl 区域、diff 区域和 extra 区域。

步骤 3　解压缩并完成如图 3 所示的 Add 操作。根据解压缩 ctrl 区域得到 c_i 中的数据 (x_i,y_i,z_i),根据解压缩 diff 区域得到大小为 x_i 个字节的数据块 d_i,将 d_i 存放于辅助空间 a 中,并依据 c_i 中的偏移量 z_i

提取出 old 文件中与 d_i 对应的数据,将其存放于辅助空间 b 中。将上述 2 个数据块相加后得到目标数据,并以追加的方式暂存在辅助空间 a 中。

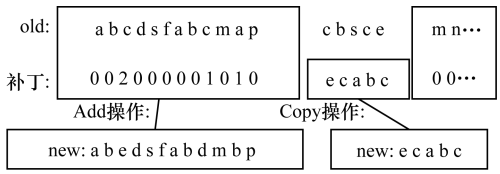

图 3 bsdiff 算法的解码

Fig. 3 Decoding of bsdiff algorithm

步骤 4 解压缩并完成如图 3 所示的 Copy 操作。解压缩 extra 区域得到大小为 y_i 个字节的数据块 e_i,将 e_i 中的数据以追加的方式复制到辅助空间 a 中。

步骤 5 调整负责管理辅助空间 a 与辅助空间 b 的指针,循环执行步骤 3 与步骤 4,直至应用完补丁文件中所有的 c_i 数据,辅助空间 a 中的数据即为原 new 文件中的数据。

步骤 6 将辅助空间 a 中的数据全部写入 new 文件,释放申请的辅助空间,并关闭打开的解压缩进程。

1.2 patch 过程内存消耗较大的原因分析

首先,bsdiff 算法在 patch 过程中,采用并行解压的方式解压缩补丁文件,这是造成 bsdiff 算法内存消耗较大的原因之一,因为同时解压 3 个部分需要的解压内存几乎是解压一个部分的三倍[9]。其次,在 patch 过程的步骤 1 中申请的辅助空间过大,这是由于多数情况下 Add 操作与 Copy 操作并不需要与新版本固件、旧版本固件同等大小的辅助空间。最后,bsdiff 算法内部使用压缩级别为 9 的 bzip2 压缩算法压缩补丁文件,但是,bzip2 压缩算法的解压缩过程需要大量的运行内存,并不适用于在资源有限的嵌入式设备上使用[12]。

2 MS-bsdiff 增量更新算法

2.1 MS-bsdiff 算法的补丁文件格式

本文采用文献[9]中改进的补丁文件格式,具体如图 4 所示。将每次应用 c_i 构建新版本固件所需的 d_i 数据块、e_i 数据块按序存放在补丁文件中,执行 patch 过程只需要依次找到 c_i 所需的 d_i 数据块和 e_i 数据块即可逐步构建新版本固件。改进后的文件格式不需要频繁记录并计算每次应用 c_i 所需的 d_i 数据块、e_i 数据块在补丁文件中的偏移量,因此能够节省计算量和内存消耗。

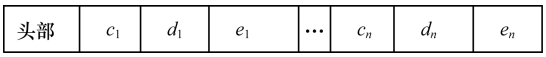

图 4 改进的补丁文件格式

Fig. 4 Improved patch file format

2.2 MS-bsdiff 算法

MS-bsdiff 算法的 diff 过程主要涉及 2 个过程,分别为编码生成临时文件与压缩临时文件生成待传输的补丁文件,其具体步骤为:

步骤 1 创建空的临时文件和补丁文件。

步骤 2 通过后缀排序以及二分查找算法检索 old、new 文件中的相似匹配区域和不匹配区域。

步骤 3 计算得到当前相似匹配区域和不匹配区域对应的 d_i 数据块与 e_i 数据块,以及生成其对应的控制信息 c_i,将 c_i、d_i、e_i 按顺序依次写入临时文件。

步骤 4 重复步骤 2 与步骤 3,直至 new 文件的末尾。

步骤 5 将算法的版本信息和 new 文件大小写入临时文件的头部区域,并对临时文件执行一次整体的压缩,从而得到待传输的补丁文件。

MS-bsdiff 算法的 patch 过程主要涉及解压缩补丁文件得到临时文件,以及应用临时文件进行解码得到新版本固件 2 个过程,其具体步骤为:

步骤 1 对补丁文件执行一次串行解压,得到临时文件。

步骤 2 创建空的 new 文件,申请大小为 m 的 2 个辅助空间 $a_1[m]$、$a_2[m]$,申请大小为 n 的辅助空间 $b_1[n]$。

步骤 3 取出临时文件中的 c_i 指令,并存放于辅助空间 b_1 中,根据 c_i 中的信息执行 Add 操作。判断 c_i 中代表 d_i 数据块大小的 $x_i \leqslant m$ 是否成立,若成立,则将 d_i 数据块从补丁文件中取出存放于辅助空间 a_1 中,并依据偏移量 z_i 从旧版本固件中取出与 d_i 数据块对应的数据,存放于辅助空间 a_2 中,然后将辅助空间 a_1 和辅助空间 a_2 中的数据相加,将得到的目标数据直接写入 new 文件;若 $x_i \leqslant m$ 不成立,则分批执行本次 Add 操作,即每次从补丁文件、旧版本固件中分别取出大小为 m 的数据块存放在辅助空间 a_1 与辅助空间 a_2 中,将辅助空间 a_1 和辅助空间 a_2 中的数据相加得到的目标数据写入 new 文件中,并记录已经处理的大小为 p 的数据块,直至 p 等于 x_i 时,完成了对 d_i 数据块的处理。

步骤 4 根据 c_i 中的信息执行 Copy 操作。判断 c_i 中代表 e_i 数据块大小的 $y_i \leqslant m$ 是否成立,若成立,

则将 e_i 数据块从补丁文件中取出并存放于辅助空间 a_1 中,并将辅助空间 a_1 中的数据写入 new 文件中;否则与步骤 3 类似,分批处理 e_i 数据块,直至将 e_i 数据块中的内容全部写入 new 文件。

步骤 5　重复执行步骤 3 与步骤 4,直至应用完临时文件中所有的 c_i 指令,从而完成对 new 文件的写入,并释放申请的辅助空间,删除临时文件。

由上述步骤可知,MS-bsdiff 算法的解码过程需要的辅助空间大小的固定值为 $2m+n$,其中,m 和 n 小于新、旧版本固件的大小,m 可以根据嵌入式设备的实际内存大小进行调整,n 与 c_i 的大小相同。此外,虽然 MS-bsdiff 算法在 patch 过程需要一个临时文件,占用了设备端的部分存储空间,但存储空间通常远大于内存空间,而且存储空间相比于内存空间更易扩展,因此,对于内存空间受限的嵌入式设备,可以消耗部分存储空间以满足程序运行时的内存需求。

2.3　非对称的无损压缩算法

在无损压缩算法领域,LZ(Lempel-Ziv)族算法在压缩和解压缩过程中的时间和内存需求方面表现出高度的不对称性[13]。本文将表 1 列出的 6 种无损压缩算法应用于 MS-bsdiff 算法的压缩与解压缩过程,且这 6 种无损压缩算法都实现了对应的处理文件流的函数接口,除了 bzip2 算法以外,其他算法都具备 LZ 族算法非对称的特点。同时,为了获得更好的压缩效果,各种算法都选择当前版本软件所支持的最大压缩级别。

表 1　6 种无损压缩算法描述

Table 1　Description of six lossless compression algorithms

算法名称	软件版本	压缩等级
bzip2 算法	bzip2-1.06	9
Lzw 算法	Lzw-0.1.1	4
Lzo 算法	Lzop-2.10	9
fastLZ 算法	fastLZ-1.1.0	2
LZ4 算法	Lz4-dev-1.8.3	9
Lzma 算法	Liblzma-5.1.0	9

3　实验与结果分析

实验以机智云物联网科技公司为 ESP8266 芯片开发的开源固件为样本,其具体说明如表 2 所示,其中,固件的名称为版本号加上该固件对应的处理器类型,以 25_16.bin 为例说明,它表示该固件的版本号为 25,对应的处理器类型为 16 位。

表 2　实验样本数据集

Table 2　Experimental sample dataset

编号	旧版本固件		新版本固件	
	名称	大小/Byte	名称	大小/Byte
1	25_16.bin	2 093 056	28_16.bin	2 097 152
2	28_16.bin	2 097 152	29_16.bin	2 093 056
3	29_16.bin	2 093 056	34_16.bin	2 097 152
4	25_32.bin	4 190 208	28_32.bin	4 194 304
5	28_32.bin	4 194 304	29_32.bin	4 190 208
6	29_32.bin	4 190 208	34_32.bin	4 194 304

3.1　评价指标与实验环境配置

3.1.1　评价指标

本文以补丁文件大小、patch 过程的内存消耗以及时间消耗作为评价增量更新算法的关键指标。补丁文件大小直接影响增量更新算法的可行性,这是由于即使是很小的新版本固件的变化都可能需要一个很大的补丁文件来记录新、旧版本固件之间的差异信息[14]。此外,已有研究表明,在增量更新过程中,设备通过无线传输接收固件的能量消耗远大于 patch 过程[15-16],因此较小的补丁文件能够节省设备的能量。patch 过程的内存消耗不宜过大,因为较大的内存消耗将对设备上执行的其他任务造成影响,甚至决定能否在设备上运行构建新版本固件的程序[17]。同时,合理的时间消耗也是需要考虑的性能指标。

本文跟踪算法的补丁文件大小以及在 patch 过程中的时间和运行内存消耗。补丁文件的大小用压缩率来衡量,压缩率 n 的计算方法为:

$$n=(s_{new}-s_{patch})/s_{new}\times 100\% \tag{1}$$

其中,s_{new} 为新版本固件的大小,s_{patch} 为补丁文件的大小。压缩率 n 反映了相较于发送整个新版本固件,发送补丁文件节省的文件大小的程度。压缩率越大,则表示补丁文件越小,越节省因传递补丁文件而造成的时间、能量和网络流量开销[18]。

3.1.2　实验环境配置

实验环境为 Intel® Core™ i5-3230M 处理器,CPU 2.60 GHz,内存 4 GB,硬盘 465 GB。运行环境为 Windows 7(32 位)系统中的 VMware 虚拟机,编程语言为 C 语言。编译器为 gcc-4.8.4.虚拟机的 linux 操作系统,版本为 ubuntu-14.04.5,配置参数为单处理器,内存为 1 GB,硬盘为 20 GB。

3.2　实验设计与分析

实验选择 MS-bsdiff 算法的 patch 过程中的 m = 32 KB,n = 24 Byte。当使用相同的实验样本时,MS-bsdiff 算法中 patch 过程的解码任务处理的数

据相同,需要的辅助空间的固定大小为 $2m+n$,因此 MS-bsdiff 算法采用不同的压缩算法时,patch 过程的时间和内存消耗主要取决于解压缩任务的时间和内存消耗。实验分别测试了 MS-bsdiff 算法 patch 过程的解压缩与解码任务的时间和内存消耗,由于解压缩与解码任务是先后执行的,因此 patch 时间是解压缩时间与解码时间之和,patch 内存是解压缩内存与解码内存中的较大值。利用基于 bsdiff-4.3 版本的开源代码实现 MS-bsdiff 算法,并将表 1 中列出的 6 种压缩算法应用于 MS-bsdiff 算法的压缩、解压缩过程。本文选取了当前普遍使用的 xdelta 算法[19]、vcdiff 算法[20]、zdelta 算法[21] 3 种固件增量更新算法与 MS-bsdiff 算法进行性能比较,结果如表 3 所示。

表 3 各种增量更新算法的性能比较

Table 3 Performance comparison of various incremental update algorithms

算法	压缩率/%	时间/s			运行内存/KB		
		解压缩时间	解码时间	patch 时间	解压缩内存	解码内存	patch 内存
xdelta-1.1.3 算法	96.36	—	0.070	0.070	—	4 090.7	4 090.7
vcdiff-3.0.7 算法	96.62	—	0.170	0.170	—	8 549.3	8 549.3
zdelta-2.1.0 算法	97.69	—	0.192	0.192	—	8 538.6	8 538.6
bsdiff-4.3.0 算法	98.37	—	0.137	0.137	—	8 095.3	8 095.3
MS-bsdiff(bzip2)算法	98.26	0.022	0.120	0.142	2 125.0	1 358.7	2 125.0
MS-bsdiff(Lzw)算法	97.82	0.240	0.120	0.360	1 200.0	1 358.7	1 358.7
MS-bsdiff(Lzo)算法	97.48	0.080	0.120	0.200	1 528.0	1 358.7	1 528.0
MS-bsdiff(fastLZ)算法	96.84	0.370	0.120	0.490	1 309.0	1 358.7	1 358.7
MS-bsdiff(LZ4)算法	97.54	0.068	0.120	0.188	5 318.7	1 358.7	5 318.7
MS-bsdiff(Lzma)算法	98.43	0.070	0.120	0.190	7 048.7	1 358.7	7 048.7

从表 3 可以看出,MS-bsdiff 算法具有良好的压缩性能。当 MS-bsdiff 算法内部使用 fastLZ 算法时,其压缩率最小为 96.84%,比 xdelta-1.1.3 算法与 vcdiff-3.0.7 算法的压缩率大。此外,MS-bsdiff 算法的 patch 过程的时间相比于 bsdiff 算法有所增长,但是其 patch 过程的内存均小于 bsdiff 算法。当 MS-bsdiff 算法采用与 bsdiff 算法相同的压缩算法 bzip2 时,其压缩率为 98.26%,相比于 bsdiff 算法有所下降,这主要是因为 bzip2 算法应用于高度结构化的数据时压缩效果更好[5],MS-bsdiff 算法使用新的补丁文件格式,影响了 bzip2 的压缩性能。bzip2 算法的串行解压内存为 2 125.0 KB,bsdiff 算法的 patch 内存为 8 095.3 KB,这说明 bsdiff 算法中负责解压缩补丁文件的 3 个并行的 bzip2 解压进程,是造成 bsdiff 算法内存消耗较大的主要原因。当 MS-bsdiff 算法内部使用 Lzma 算法时,其压缩率最高为 98.43%,且其 patch 过程的内存相比于 bsdiff 算法也有所下降。当 MS-bsdiff 算法内部使用 Lzw 算法时,其压缩率为 97.82%,均大于 xdelta-1.1.3 算法、vcdiff-3.0.7 算法和 zdelta-2.1.0 算法的压缩率,同时,其解压内存仅为 1 200.0 KB,小于使用 bzip2 算法时的解压内存 2 125.0 KB,在内存资源受限的设备中,MS-bsdiff 算法使用 Lzw 算法进行压缩与解压缩时,可以进一步降低设备端 patch 过程的内存消耗。

应用表 2 中的 6 组实验数据,得到 MS-bsdiff 算法内部使用 Lzw 算法时的 patch 运行内存与 bsdiff 算法的 patch 运行内存比较,如图 5 所示。从图 5 可以看出,当 MS-bsdiff 算法内部使用 Lzw 算法时,其 patch 过程的运行内存远小于 bsdiff 算法,且其 patch 过程的运行内存几乎是恒定的。因此,当采用内部使用 Lzw 算法的 MS-bsdiff 算法执行固件增量更新时,其 patch 过程不会由于复杂的系统功能导致的固件体积增大,而需要更多的内存空间。

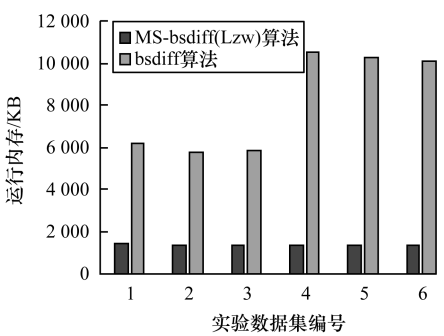

图 5 MS-bsdiff(Lzw)算法与 bsdiff 算法的 patch 过程运行内存比较

Fig. 5 Comparison of running memory between MS-bsdiff(Lzw) algorithm and bsdiff algorithm in patch process

4　结束语

在 bsdiff 算法的基础上,本文提出一种节约内存的增量更新算法 MS-bsdiff,并将非对称的无损压缩算法应用到该算法的压缩与解压缩过程。实验结果表明,在该算法中使用 Lzw 算法进行压缩与解压缩时,其 patch 过程的内存消耗约为 bsdiff 算法的 1/6,而且内存消耗不会因固件体积的增大而增大,同时该算法能够生成较小的补丁文件。下一步将继续探索压缩效率高、解压缩内存消耗小的无损压缩算法在 MS-bsdiff 算法中的性能,以提高增量更新算法的效率。

参考文献

[1] HUYNH-VAN D, TRAN-QUOC K, LE-TRUNG Q. An empirical study on approaches of internet of things reconfiguration[C]//Proceedings of the 7th International Conference on Communications and Electronics. Washington D.C., USA:IEEE Press,2018:57-62.

[2] DU Zhenlong, SHA Guangxia, LI Xiaoli, et al. Customization and update research on capsule type firmware based on UEFI[J]. Computer Engineering, 2014,40(10):292-295,303. (in Chinese)
杜振龙,沙光侠,李晓丽,等.基于 UEFI 的胶囊式固件定制更新研究[J].计算机工程,2014,40(10):292-295,303.

[3] KACHMAN O, BALAZ M. Effective over-the-air reprogramming for low-power devices in cyber-physical systems[M]. New York, USA: Springer International Publishing,2016:284-292.

[4] KACHMAN O,BALAZ M,MALIK P. Universal framework for remote firmware updates of low-power devices[J]. Computer Communications,2019,139:91-102.

[5] PERCIVAL C. Matching with mismatches and assorted applications[D]. Oxford,UK:University of Oxford,2006.

[6] XIA Qi. Research on difference algorithm in mobile Internet data updating[D]. Chengdu:University of Electronic Science and Technology of China,2014. (in Chinese)
夏棋.移动互联网增量数据差分更新算法研究[D].成都:电子科技大学,2014.

[7] SHI Chao. The research and design of incremental OTA upgrade system based on Android platform[D]. Zhenjiang:Jiangsu University,2017. (in Chinese)
施超.基于 Android 平台 OTA 增量升级系统研究与设计[D].镇江:江苏大学,2017.

[8] NAKANISHI T, SHIH H H, HISAZUMI K, et al. A software update scheme by airwaves for automotive equipment[C]//Proceedings of 2013 International Conference on Informatics, Electronics and Vision. Washington D.C.,USA:IEEE Press,2013:1-6.

[9] TERAOKA H, NAKAHARA F, KUROSAWA K. Incremental update method for resource-constrained in-vehicle ECUs[C]//Proceedings of the 5th Global Conference on Consumer Electronics. Washington D.C., USA:IEEE Press,2016:1-2.

[10] LI Jinfeng. Design and implementation of APP hosting platform based on play framework[D]. Beijing:Beijing University of Posts and Telecommunications,2018. (in Chinese)
李锦峰.基于 play framework 的 APP 托管平台的设计与实现[D].北京:北京邮电大学,2018.

[11] MOTTA G, GUSTAFSON J, CHEN S. Differential compression of executable code[C]//Proceedings of 2007 Data Compression Conference. Washington D.C., USA:IEEE Press,2007:103-112.

[12] KOMANO Y,XIA Z F,KAWABATA T,et al. Efficient and secure firmware update/rollback method for vehicular devices[C]//Proceedings of Information Security Practice and Experience. Berlin, Germany: Springer,2018:455-467.

[13] FERREIRA A J,OLIVEIRA A L,FIGUEIREDO M A T. On the suitability of suffix arrays for lempel-ziv data compression[C]//Proceedings of International Conference on E-Business and Telecommunications. Berlin, Germany: Springer,2008:267-280.

[14] WEI Min,WANG Yi. Research and application of remote update technology of IoT terminal[J]. Telecommunications Science,2018,34(10):137-142. (in Chinese)
魏民,王艺.物联网云平台终端远程更新技术研究与应用[J].电信科学,2018,34(10):137-142.

[15] MAZUMDER B,HALLSTROM J O. An efficient code update solution for wireless sensor network reprogramming[C]//Proceedings of International Conference on Embedded Software. Washington D.C.,USA:IEEE Press,2013:1-10.

[16] STOJKOSKA B R,NIKOLOVSKI Z. Data compression for energy efficient IoT solutions[C]//Proceedings of the 25th Telecommunication Forum. Washington D.C., USA:IEEE Press,2017:1-4.

[17] STOLIKJ M, CUIJPERS P J L, LUKKIEN J J. Efficient reprogramming of wireless sensor networks using incremental updates[C]//Proceedings of 2013 IEEE International Conference on Pervasive Computing and Communications Workshops. Washington D.C., USA:IEEE Press,2013:584-589.

[18] CHENG Zhilong, NI Guiqiang, JIANG Jinsong, et al. Incremental update algorithm based on dynamic dictionary[J]. Journal of PLA University of Science and Technology (Natural Science Edition), 2015, 16(5): 426-432. (in Chinese)
陈志龙,倪桂强,姜劲松,等.基于动态字典的增量更新算法[J].解放军理工大学学报(自然科学版),2015,16(5):426-432.

[19] MACDONALD J. File system support for delta compression[D]. Berkeley, USA: University of California at Berkeley,2000.

[20] KORN D, MACDONALD J, MOGUL J, et al. The VCDIFF generic differencing and compression data format[EB/OL]. [2019-09-20]. https://www.rfc-editor.org/in-notes/pdfrfc/rfc3284.txt.pdf.

[21] TRENDAFIL O V D, MEMON N, SUEL T. Zdelta: an efficient delta compression tool[EB/OL]. [2019-09-20]. https://www.researchgate.net/profile/Nasir_Memon/publication/2556720_zdelta_An_Efficient_Delta_Compression_Tool/links/0046351f9ae28a19bd000000/zdelta-An-Efficient-Delta-Compression-Tool.pdf.

编辑　刘继娟