第四章 大话文件传输

在不限于嵌入式的很多工业应用场合，文件传输的是非常常见的需求。小到上位机与单片机之间的烧录过程，大到互联网中从服务器下载文件，再大到地面站到卫星，甚至深空探测器之间的资料传输，比如珍贵的火星照片。文件传输的根本是协议的设计与实现。而设计一个稳定可靠高效鲁棒的文件传输协议，是需要一些技巧和智慧的。本章将详细介绍一些传输协议，最终大家会发现其实它们的基本思想是一样的。

1、Xmodem 协议族

说到文件传输协议，就必须要说 Xmodem，它是所有文件传输协议的鼻祖。1978 年，一位 IBM 的工程师尝试使用调制解调器来进行文件数据传输，并使其有一定的纠错能力，从而创建了一套协议，命名为 Xmodem。其基本思想是：发送数据包大小为 128 字节。如果包成功接收，接收方会返回一个肯定应答信号（ACK），如果发现错误，则返回一个否定应答信号（NAK）并重新发送数据包。Xmodem 最初使用奇偶校验作为查错控制的方法。

1.1 Xmodem 的传输过程

其实 Xmodem 的逻辑很简单，如图 4.1 所示。

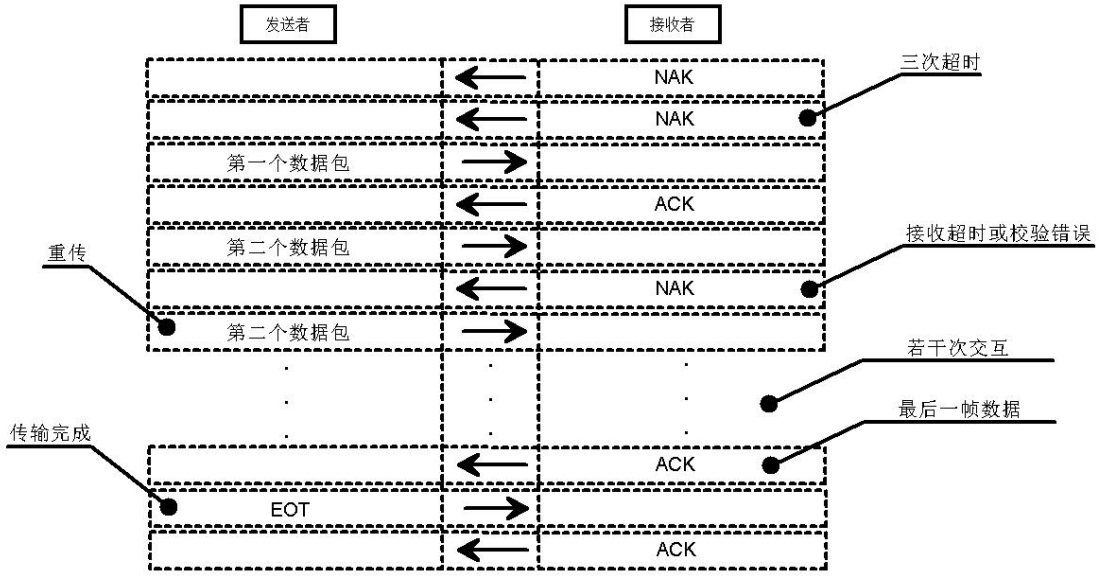

图 4.1 Xmodem 的文件传输过程示意

文件数据的传输过程其实就是收发双方交互协商的过程。既然是协商就会有双方遵循的约定，即所谓的协议。

首先是控制字的定义：

<SOH> 01H

<EOT> 04H

<ACK> 06H

<NAK> 15H

<CAN> 18H

图 4.1 中已经涉及到了 ACK、NAK 和 EOT，不再赘述。关于 CAN 的作用是这样的：当接收方发送 CAN 表示无条件结束本次传输过程，发送方收到 CAN 后，无需发送 EOT 来确认，直接停止数据的发送。

然后就是数据包的定义：

SOH	包号	包号(反码)	128字节数据	校验和

SOH 即 Start of Heading，包头开始字符；随后第 2 个字节是包号，它在传输中是依次递增的，需要注意的是，第一包数据的包号从 1 开始，然后 255->0 循环往复；第 3 个字节是包号的按位取反；4~131 字节是 128 字节的文件分片数据；最后 1 个字节是校验和，即 128 字节数据按字节连加之和。

可以看到，Xmodem 每一帧数据都有独立的数据校验，加之它对重传的支持，使得文件传输的正确性得以保障。一般来说，只要通过 Xmodem 传输成功的文件，数据上是不会出现差错的，也就不需要单独再进行额外的文件校验(但是实际上我们还是会对接收到的文件整体进行检查，比如通过文件自身的校验码，因为除了传输，差错还可能出现在其它方面，比如存储)。

以上所介绍的内容其实是 Xmodem 最原始的形态，在此基础上它又有所改进，甚至衍生出一套协议族。

最直接的一个改进，就是将其数据包的校验和，改为了双字节的 CRC，实际是换汤不换药，但是它的传输交互过程有一些变化，如图 4.2。

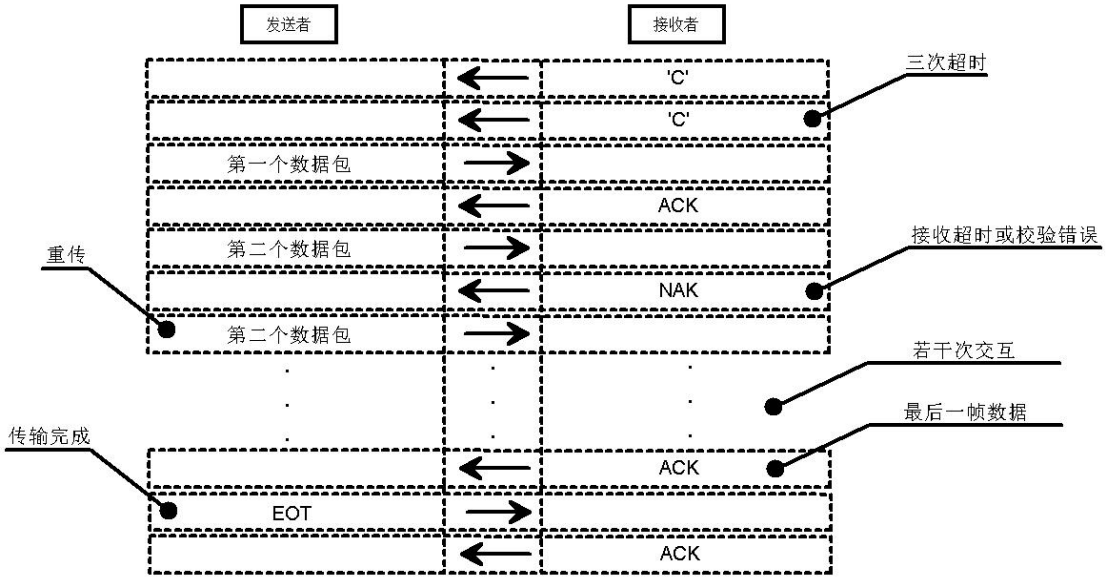

图 4.2 Xmodem(CRC)的文件传输过程示意

可以看到，Xmodem 如果使用 CRC 的话，在传输之初接收者是使用字符'C'来进行握手的，而校验和方式下，则是 NAK，即 0X15。Xmodem 的这一改进，并不是对原来校验和方式的取代，而是补充。也就是说，现行实际 Xmodem 发送者有可能是采用 CRC 的，也可能是校验和，或者是两者都支持。

这样问题就来了：接收者是用'C'来握手，还是用 NAK 呢？它决定了数据包的校验方式。

接收者一般是设备端，为了使其更加通用（不挑协议），通常我们会兼顾校验和与 CRC 两种方式，具体实现方法如图 4.3。

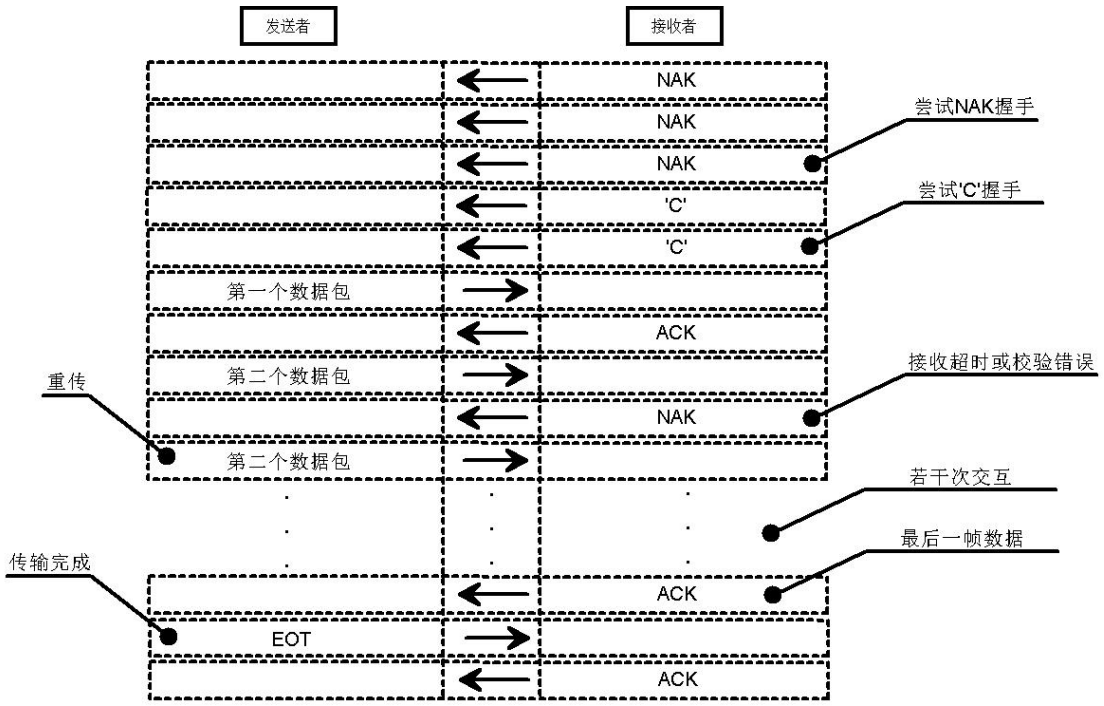

图 4.3 Xmodem(兼顾校验和与 CRC)的文件传输过程示意

除了 Xmodem(CRC)，Xmodem 的改进还有 Xmodem-1K，顾名思义，它的数据包中分片数据长度为 1KB，目的是提高传输效率。

1.2 Ymodem 的传输过程

几经改进的 Xmodem，人们给了它一个新的名字，Ymodem。它更加完备而高效，传输过程如图 4.4。

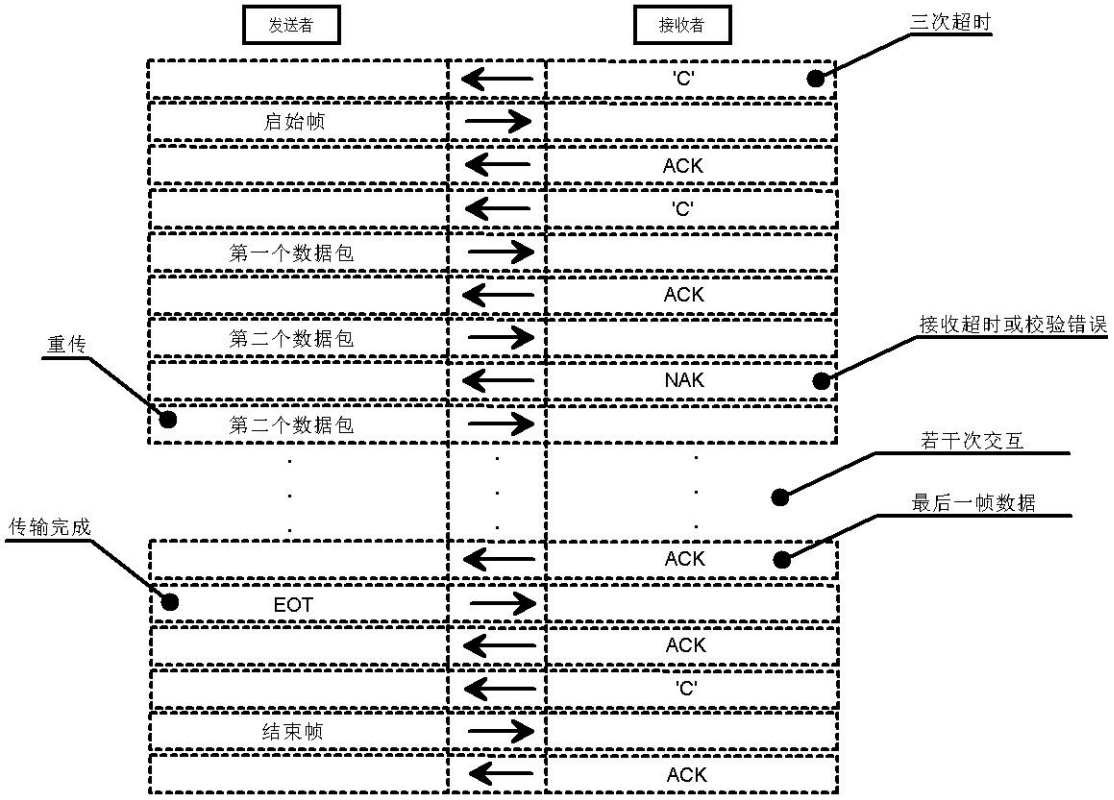

图 4.4 Ymodem 的文件传输过程示意

可以看到，Ymodem 的传输过程略显复杂，它最显著的特性是增加了启始帧和结束帧，两者的定义如图 4.5。

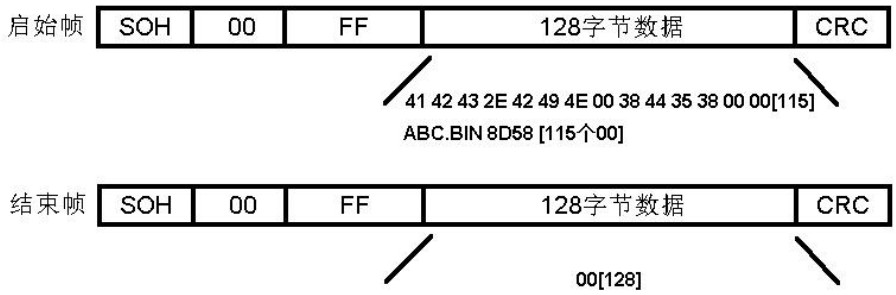

图 4.5 Ymodem 的启始帧与结束帧

数据包中分片数据长度为 1KB，其格式定义与 Xmodem 不同，如图 4.6 所示，它引入了一个新的控制字，即 STX，它是数据包的开始字符，值为 0X02。

图 4.6 Ymodem 的数据帧定义

所以，Ymodem 中有两种不同长度的数据帧，以 SOH 开头的数据帧数据长度为 128 字节，以 STX 开头的数据帧数据长度为 1024 字节。其中启始帧描述了更多的文件信息，包括文件名与文件大小。这一点是比 Xmodem 要完善的。同时，我们注意到 Ymodem 是只支持 CRC 校验的，这比 Xmodem 更标准更统一。

Ymodem 是现行 Xmodem 协议族中最流行的版本，在很多的 Bootloader、OTA 等嵌入式方案中都有所应用，比如 RT Trhead 中就用 Ymodem 来实现上位机向其 DFS(设备文件系统)的文件传输。

正所谓改进之心永远不死，人们对 Ymodem 也不乏改进，衍生出了 Ymodem-G 版本，它去掉了 CRC 校验，当然这样就不能保证文件传输的正确性了。我也不知道这样算是“改进”还是“改退”，也许它有特定的应用场景吧。

1.3 关于 Zmodem

前面介绍了 Xmodem 与 Ymodem，它们有一定的继承性，但是 Zmodem 与它们相比完全不是一个量级的东西，要复杂得多。它不是基于发送与应答的机制，而是类似流式传输的一种协议。关于 Zmodem 我也没有找到比较明确的协议资料，所以并没有深入去研究它。这里我作了一个实验，抓取 Zmodem 实际传输过程中的收发双方的原始串口数据，贴在下面，您若有兴趣可以自行研究一下，如图。

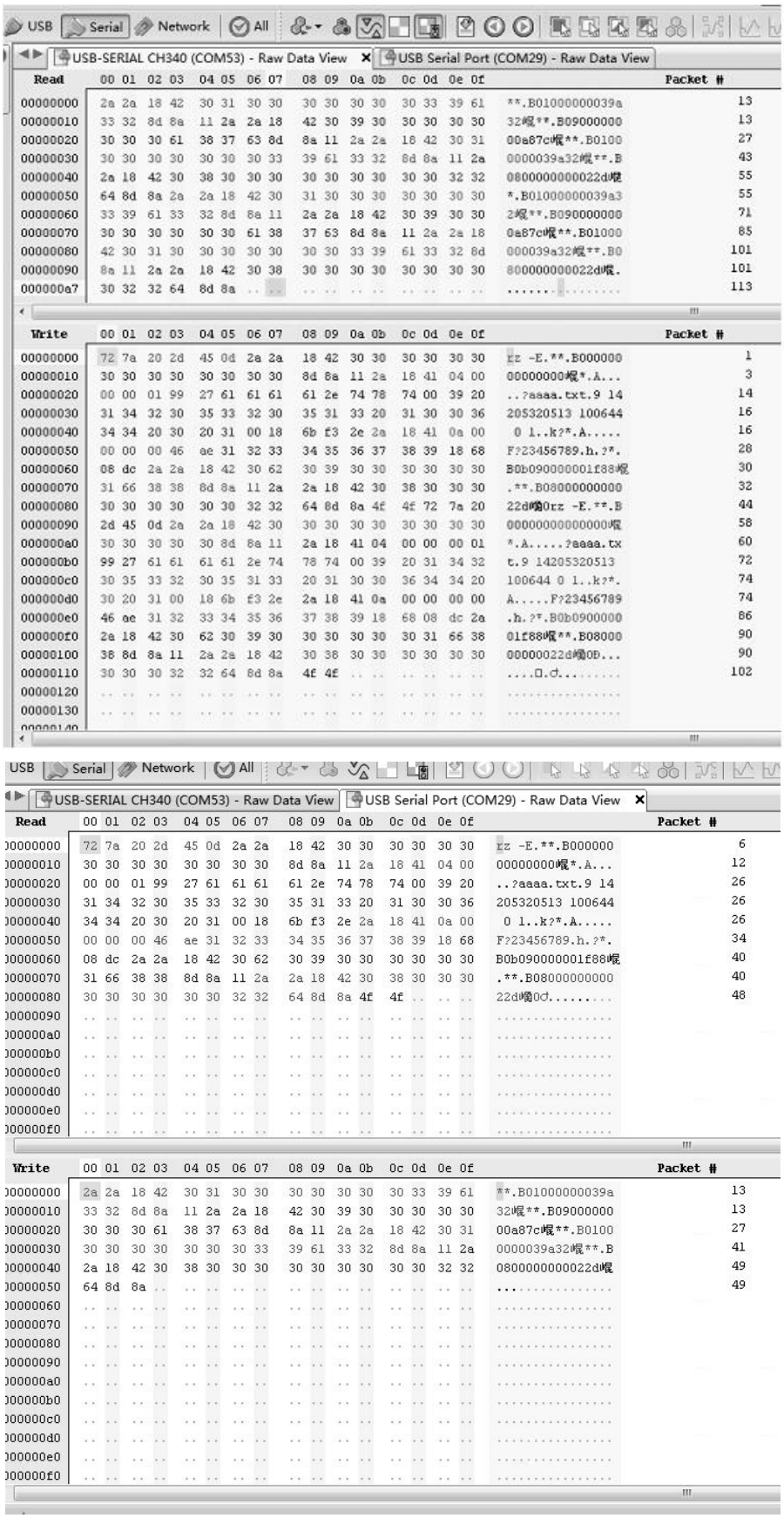

图 4.6 Zmodem 文件传输收发双方原始串口数据(上发下收)

说明：使用 Xshell 打开两个串口(这两个串口收发相连)，使用其中一个串口通过 Zmodem 发送文件 aaaa.txt（此文件内容是 123456789），另一个串口接收文件。此过程中，使用串口监控软件记录其原始数据，以待分析。

“Xshell 是什么？串口监控软件是哪个？”关于这些工具软件，振南专业安排了一章来进行集中讲解，请关注相应章节。

1.4 AVRUBD 的传输过程

看过《深入浅出话 Bootloader》一章的读者就会知道，AVRUBD 是网友 shaoziyang 在实现 AVR 单片机的通用 Bootloader 时所使用的文件传输协议。严格来说，AVRUBD 并不能算是 Xmodem 协议族中的一员，它只是对 Xmodem 的应用，只不过有些改进，请看图 4.7。

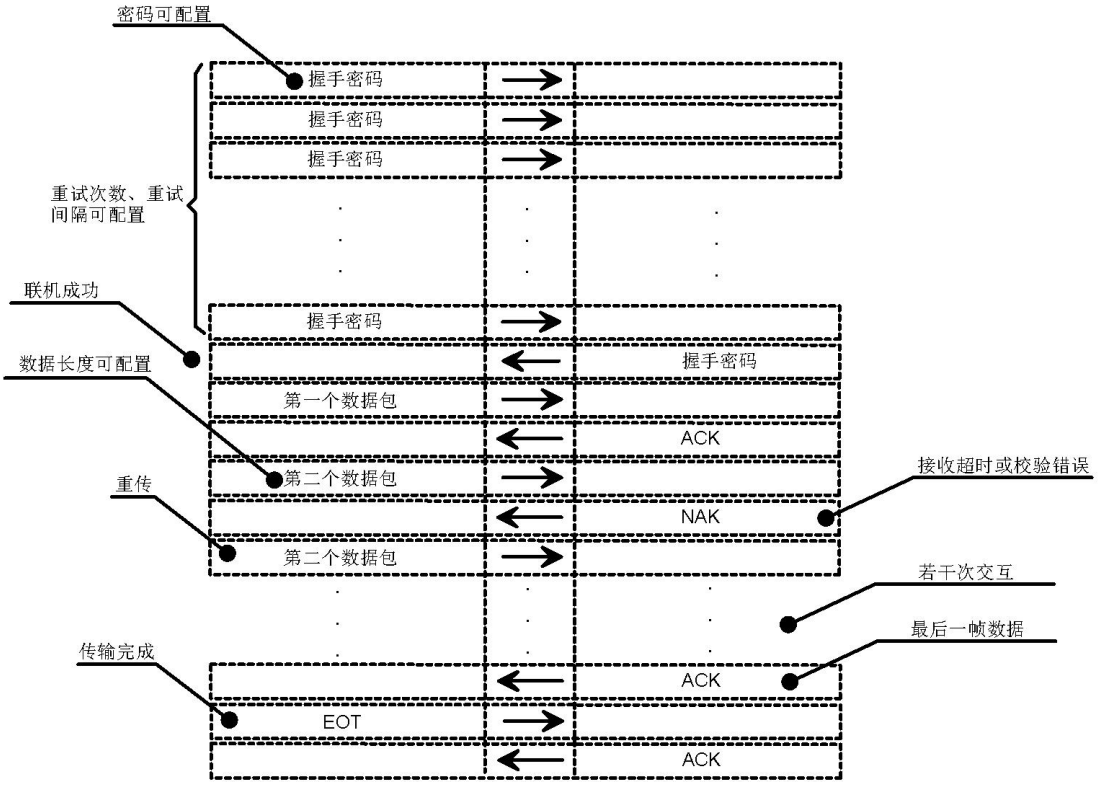

图 4.7 AVRUBD 传输文件的过程

可以看到，AVRUBD 的主要改进点有：

1、增加了密码联机的机制；

2、数据包中的数据长度可配置，不再局限于 128B 或 1KB，而可以更灵活地配置；

这些改进点都是为单片机烧录程序服务的，1、防止随意烧录，相当于增加了权限控制；2、考虑单片机实际内存容量和烧录效率，可以灵活配置数据长度。

关于 AVRUBD 的更多内容在《Bootloader》一章中已经有详细描述，包括上位机软件、蓝牙隔空文件传输等，这里不再赘述。

2、更多的文件传输协议

Xmodem 协议族通常都是通过串口来进行较近距离的文件传输，但实际上很多文件传输的需求远比这要复杂，需要我们灵活的针对特定场景设计通信协议来实现更加多样化的文件传输。

2.1 振南的 CAN 文件传输

通过 CAN 来进行文件传输源自于我的“共享充电柜”项目，如图 4.8。

图 4.8 共享充电柜实物外观及内部电路

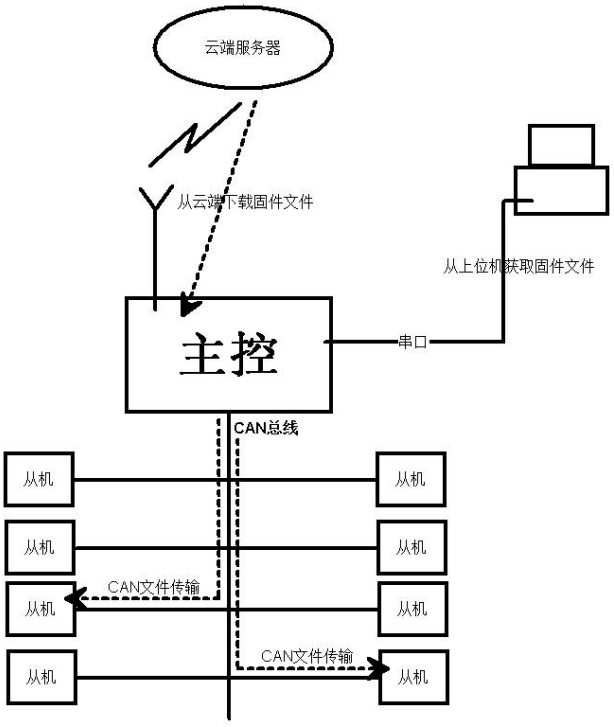

图 4.9 共享充电箱的整体示意图

它包括一个主控和若干个从机，它们之间通过 CAN 总线连接以实现通信。主控具有 4G 通信功能，可接收云端下发的命令，比如打开柜、开始充电、结束充电等，也可以向云端上传状态信息，如柜门开关状态、电流功率等。关于这个项目更详细的内容请关注本书相应章节。

对于主控及从机的固件升级，也都实现了远程更新。主控我们暂且不提，先来说一下从机的固件烧录：主控从云端下载或者从上位机获取固件文件(从机固件)，然后通过

CAN 总线将此固件再进一步传给某个从机(通过 CAN ID 进行区分)，最终通过从机的 Bootloader 完成烧录。

在这个过程中，CAN 文件传输协议是关键，如图 4.10。

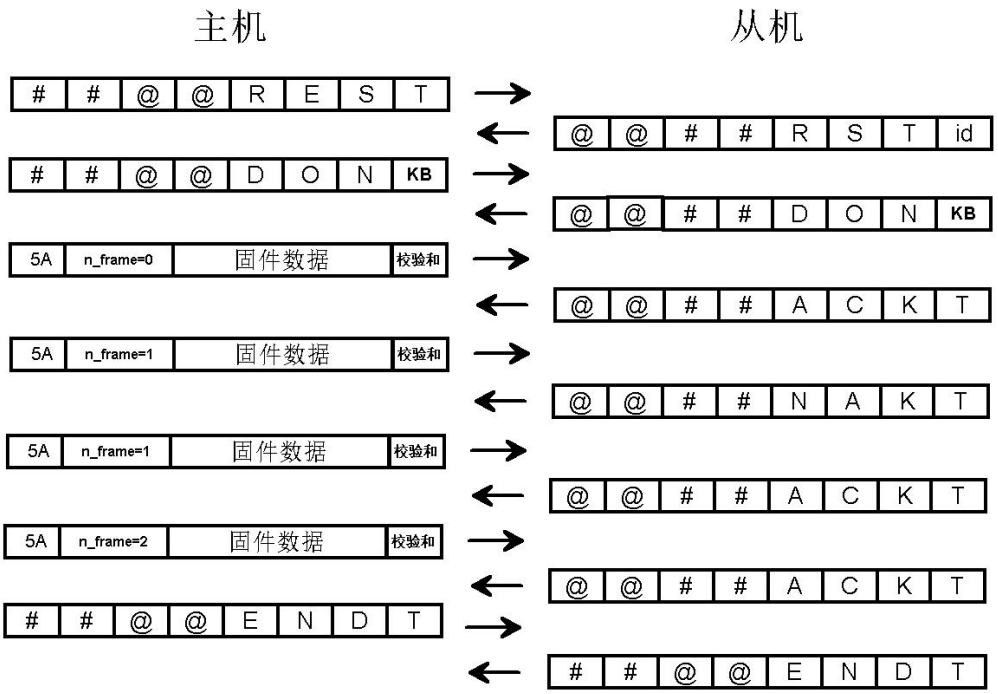

图 4.10 通过 CAN 进行文件传输的协议示意

其实 CAN 总线并不适合于批量的数据传输，它每一帧数据仅能传输 8 个字节。但为了实现高度自动化的批量烧录，我还是基于 CAN 总线实现了文件传输。仔细看图 4.10 中的传输过程，你就会发现它与 Xmodem 其实大同小异，实现思路是一样的。

“相比于 CAN 文件传输，我更想知道你这个主控是如何实现从云端下载固件的，也就是嵌入式网络文件传输是如何实现的？”别急，下一节就会涉及到这方面的内容。我上面以“智能充电柜”这个项目为例，也是为了引出这个问题。

2.2 通过 HTTP 下载文件

。。。。。。

2.3 Json 传输文件

有经验的嵌入式工程师对 Json 并不陌生，我们可以用它来实现不同平台之间的结构化数据的交换。所以，在有些应用中 Json 会被用来进行文件数据的传输，比如我曾经研发过的一些 WIFI 设备，请看图 4.11 所示。

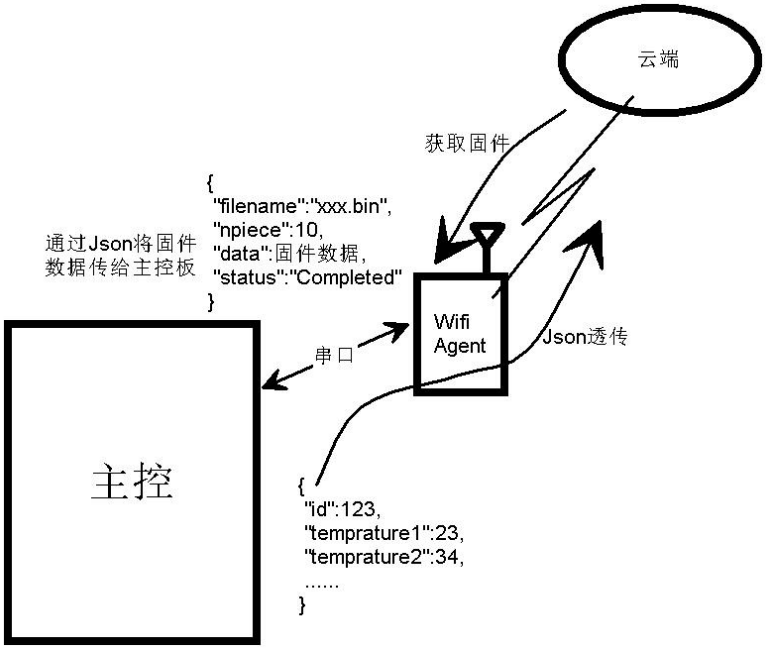

图 4.11 通过 Json 传输固件数据

我们知道 Json 本身是一个字符串，所以它是不能直接传输二进制数据的，那固件数据如何通过 Json 来传输呢？这不光是 Json 的问题，也是所有文本协议所面临的问题，比如我们经常使用的 Email 的通信协议。

核心的问题是如何基于字符来进行二进制数据的传输？为了解决这个问题，人们提出了 Base64 编码，请看图 4.12，希望大家可以体会到它的巧妙。

索引	对应字符	索引	对应字符	索引	对应字符	索引	对应字符
0	A	17	R	34	i	51	z
1	B	18	S	35	j	52	0
2	C	19	T	36	k	53	1
3	D	20	U	37	l	54	2
4	E	21	V	38	m	55	3
5	F	22	W	39	n	56	4
6	G	23	X	40	o	57	5
7	H	24	Y	41	p	58	6
8	I	25	Z	42	q	59	7
9	J	26	a	43	r	60	8
10	K	27	b	44	s	61	9
11	L	28	c	45	t	62	+
12	M	29	d	46	u	63	/
13	N	30	e	47	v		
14	O	31	f	48	w		
15	P	32	g	49	x		
16	Q	33	h	50	y		

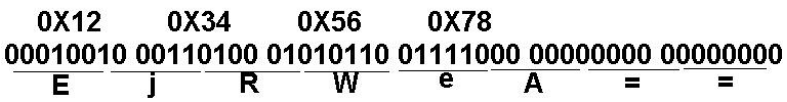

图 4.12 通过 Base64 编码来表达二进制数据

Base64 的基本思想是：将数据的二进制序列按 6 位进行分隔，然后用 64 个可打印字符进行表达。所以，每 3 个字节可以转化为 4 个字符。如果不足 3 个字节，则用=来进行占位。

本章介绍了一些文件传输的相关协议，并结合振南的几个项目对其进行了发挥拓展。这里所涉及的内容，在实际应用时可能会有更大的外延，比如 Xmodem 协议经过改良，可用于卫星数据传输。再复杂的技术也是由简单的元素构成的，了解和掌握了基础，才能让我们在技术上走得更远。希望大家从本章内容能够得到启发，并最终应用到自己的实际开发之中，让知识产生价值。