XMODEM

From Wikipedia, the free encyclopedia

XMODEM is a simple file transfer protocol developed as a quick hack by Ward Christensen for use in his 1977 **MODEM.ASM** terminal program. XMODEM became extremely popular in the early bulletin board system (BBS) market, largely because it was so simple to implement. It was also fairly inefficient, and as modem speeds increased this problem led to the development of a number of modified versions of XMODEM to improve performance or address other problems with the protocol. Chuck Forsberg collected a number of these into his YMODEM protocol, but poor implementation led to a further fracturing before they were re-unified by his later ZMODEM protocol.

Contents

XMODEM

XMODEM, like most file transfer protocols, breaks up the original data into a series of "packets" that are sent to the receiver, along with additional information allowing the receiver to determine whether that packet was correctly received.

Packet structure

The original XMODEM used a 128-byte data packet, the basic block size used on CP/M floppy disks. The packet was prefixed by a simple 3-byte header containing a <SOH> character, a "block number" from 0-255, and the "inverse" block number—255 minus the block number. Block numbering starts with 1 for the first block sent, not 0.

The packet was also suffixed with a single-byte checksum of the data bytes. The checksum was the sum of all bytes in the packet modulo 256. The modulo operation was easily computed by discarding all but the eight least significant bits of the result, or alternatively on an eight bit machine, ignoring arithmetic overflow which would produce the same effect automatically. In this way the checksum was restricted to an eight bit quantity which was able to be expressed using a single byte. For example, if the data packet contained only bytes carrying the values 130 and 130, the total of these codes is 260 and the resulting checksum is 4 using this method.

The complete packet was thus 132 bytes long, containing 128 bytes of data, for

a total throughput efficiency of about 97%.

The file was marked "complete" with a <EOT> character sent after the last block. This character was not in a packet, but sent alone as a single byte. Since the file length was not sent as part of the protocol, the last packet was padded out with a "known character" that could be dropped. In the original specification this defaulted to <SUB> or 26 decimal, which CP/M used as the end-of-file marker inside its own disk format. The standard suggested any character could be used for padding, but there was no way for it to be changed *within the protocol* itself – if an implementation changed the padding character, only clients using the same implementation would correctly interpret the new padding character.

Transfer details

Files were transferred one packet at a time. When received, the packet's checksum was calculated by the receiver and compared to the one received from the sender at the end of the packet. If the two matched, the receiver sent an <ACK> message back to the sender, which then sent the next packet in sequence. If there was a problem with the checksum, the receiver instead sent a <NAK>. If a <NAK> was received, the sender would re-send the packet, and continued to try several times, normally ten, before aborting the transfer.

A <NAK> was also sent if the receiver did not receive a valid packet within ten seconds while still expecting data due to the lack of a <EOT> character. A seven-second timeout was also used *within* a packet, guarding against dropped connections in mid-packet.

The block numbers were also examined in a simple way to check for errors. After receiving a packet successfully, the next packet should have a one-higher number. If it instead received the same block number this was not considered serious, it was implied that the <ACK> had not been received by the sender, which had then re-sent the packet.

Transfers were sender-driven, the receiver acting passively on data sent to it. However, the transfer was actually *started* by the receiver sending a single <NAK>. This was a side-effect of the way the user interacted with the sending machine, generally by navigating to the requested file, asking the sender to transfer it, then using a command in their local software to start receiving. Since the delay until the user could invoke the command was unknown, XMODEM instead made the receiver trigger the transfer when it was ready.

Minor problems

XMODEM was written for CP/M machines, and bears several marks of that operating system. Notably, files on CP/M were always multiples of 128 bytes, and their end was marked within a block with the <EOT> character. These characteristics were transplanted directly into XMODEM. However, other operating systems did not feature either of these peculiarities, and the

widespread introduction of MS-DOS in the early 1980s led to XMODEM having to be updated to notice either a <EOT> *or* <EOF> as the end-of-file marker.

For some time it was suggested that sending a <CAN> character instead of an <ACK> or <NAK> should be supported in order to easily abort the transfer from the receiving end. Likewise, a <CAN> received in place of the <SOH> indicated the sender wished to cancel the transfer. However, this character could be easily "created" via simple noise-related errors of what was meant to be an <ACK> or <NAK>. A double-<CAN> was proposed to avoid this problem, but it is not clear if this was widely implemented.

Major problems

XMODEM was designed for simplicity, without much knowledge of other file transfer protocols – which were fairly rare anyway. Due to its simplicity, there were a number of very basic errors that could cause a transfer to fail, or worse, result in an incorrect file which went unnoticed by the protocol. Most of this was due to the use of a simple checksum for error correction, which is susceptible to missing errors in the data if *two* bits are reversed, which can happen with a suitably short burst of noise. Additionally, similar damage to the header or checksum could lead to a failed transfer in cases where the data itself was undamaged.

Many authors introduced extensions to XMODEM to address these and other problems. Many asked for these extensions to be included as part of a new XMODEM standard. However, Ward Christensen refused to do this, as it was precisely the *lack* of these features, and the associated coding needed to support them, that led to XMODEM's widespread use. As he explained:

> It was a quick hack I threw together, very unplanned (like everything I do), to satisfy a personal need to communicate with some other people. ONLY the fact that it was done in 8/77, and that I put it in the public domain immediately, made it become the standard that it is...
> ...People who suggest I make SIGNIFICANT changes to the protocol, such as 'full duplex', 'multiple outstanding blocks', 'multiple destinations', etc etc don't understand that the incredible simplicity of the protocol is one of the reasons it survived.

Batch Transfers

Another problem with XMODEM was that it required the transfer to be user-driven. Typically this meant the user would navigate on the sender's system to select the file they wanted, and then invoke the transfer from their end using a command in their terminal emulator.

For automated transfers between two sites, a number of add-ons to the XMODEM protocol were implemented over time. These generally assumed the sender would continue sending file after file, with the receiver attempting to

trigger the next file by sending a <NAK> as normal at the start of a transfer. When the <NAK>'s timed out, it could be assumed that either there were no more files, or the link was broken anyway.

MODEM7

MODEM7, also known as **MODEM7 batch** or **Batch XMODEM**, was the first known extension of the XMODEM protocol. An XMODEM file transfer starts with the receiver sending a single <NAK> character to the sender, which then starts sending packets of 128-bytes of data prefixed with a <SOH>. MODEM7 changed this behavior only slightly, by sending the filename, in 8.3 filename format, before the first data packet. For a non-aware XMODEM implementation this data would simply be ignored while it waited for the <SOH> to arrive. With "aware" software, the file could be saved with that name. Transfers could continue with another <NAK>, each file being saved under the name being sent to the receiver.

TeLink

MODEM7 sent the filename as normal text, which meant it could be corrupted by the same problems that XMODEM was attempting to avoid. This led to the introduction of **TeLink** by Tom Jennings, author of the original FidoNet mailers.

TeLink avoided MODEM7's problems by standardizing a new "zero packet" containing information about the original file. This included the file's name, size, and timestamp, which were placed in a regular 128 byte XMODEM block. Whereas a normal XMODEM transfer would start with the sender sending "block 1", the TeLink header packet was labeled "block 0".

Again, a normal XMODEM implementation would simply discard the packet, the assumption being that the packet number had been corrupted. But this led to a potential time delay if the packet were discarded, as the sender could not be sure it was being <NAK>'ed because it did not understand the "block 0", or because there was a transmission error. However, TeLink was generally limited to FidoNet software, which demanded it, so it was safe to assume the receiver implemented it as well.

The basic "block 0" system became a standard in the FidoNet community, and was re-used by a number of future protocols like SEAlink and YMODEM.

XMODEM-CRC

The checksum used in the original protocol was extremely simple, and errors within the packet could go unnoticed. This led to the introduction of **XMODEM-CRC** by John Byrns,[1][2] which used a 16-bit CRC in place of the 8-bit checksum. CRC's encode not only the data in the packet, but its location

as well, allowing it to notice the bit-replacement errors that a checksum would miss. Statistically, this made the chance of detecting an error less than 16 bytes long 99.9969%, and even higher for longer data.

XMODEM-CRC was designed to be backwardly compatible with XMODEM. To do this, the receiver simply sent a `C` (capital C) character instead of a <NAK> to start the transfer. If the sender responded by sending a packet, it was assumed the sender "knew" XMODEM-CRC, and the receiver continued sending `C`'s. If no packet was forthcoming, the receiver assumed the sender did not know the protocol, and sent an <NAK> to start a "traditional" XMODEM transfer.

Unfortunately this attempt at backward compatibility had a downside. Since it was possible that the initial `C` character would be lost or corrupted, it could not be assumed that the receiver did not support XMODEM-CRC if the first attempt to trigger the transfer failed. The receiver thus tried to start the transfer three times with `C`, waiting three seconds between each attempt. This meant that if the user selected XMODEM-CRC while attempting to talk to *any* XMODEM, as it was intended, there was a potential 10 second delay before the transfer started.

To avoid the delay, the sender and receiver would generally list XMODEM-CRC separately from XMODEM, allowing the user to select "basic" XMODEM if the sender didn't explicitly list it. Ironically, any software that *did* support -CRC in their basic XMODEM transfer, as it was intended, surreptitiously suggested the user should not attempt to use -CRC. To the average user, XMODEM-CRC was essentially a "second protocol", and treated as such.

Higher throughput

Since the XMODEM protocol required the sender to stop and wait for an <ACK> or <NAK> message from the receiver, it tended to be quite slow. In the era of 300 bit/s modems, the entire 132-byte packet required just over 3.5 seconds to send (132 bytes * 8 bits per byte / 300 bits per second). If it then took 0.2 seconds for the receiver's <ACK> to make it back to the sender and the next packet to start hitting the receiver (0.1 seconds in both directions), the overall time for one packet would be 3.7 seconds, just over 92% throughput.

As modem speeds increase, the fixed delay needed to send the <ACK>/<NAK> grows in proportion to time needed to send the packet. For instance, at 2400 bit/s the packets took only 0.44 seconds to send, so if the <ACK>/<NAK> still took 0.2 seconds to make it back (this is *latency*, not throughput), the throughput has fallen to under 60%. At 9600 bit/s it is under 30% – more time is spent waiting for the reply than is needed to send the packet.

A number of new versions of XMODEM were introduced in order to address these problems. Like earlier extensions, these versions tended to be backward-compatible with the original XMODEM, and like those extensions, this led to a further fracturing of the XMODEM landscape in the user's terminal emulator.

In the end, dozens of versions of XMODEM would emerge.

SEAlink

One of the first "third party" mailers for the FidoNet system was **SEAdog**, written by the same author as the then-popular .arc data compression format. SEAdog included a wide variety of improvements, including SEAlink, an improved transfer protocol.

SEAlink used a method known as sliding windows to avoid the inter-packet delay. To do this, the protocol did not wait for the <ACK>/<NAK> to arrive, and immediately moved onto the next packet. It was only after some number of packets had been sent, the "window", that the protocol would stop and wait. In order for this to work, SEAlink needed to known *which* packet the receiver was <ACK>/<NAK>ing, which it did by appending the packet number to the <ACK> or <NAK> character.

SEAlink later added a number of other improvements, and was generally a useful protocol. However it remained rare, and was typically only found in FidoNet mailers.

XMODEM-1K

Another way to solve the throughput problem is to increase the packet size. Although the fundamental problem of latency remains, the speed at which it becomes a problem is higher. XMODEM-1K with 1024-byte packets was the most popular such solution. In this case, the throughput at 9600 bit/s is 81%, given the same assumptions as above.

XMODEM-1K was an expanded version of XMODEM-CRC, which indicated the longer block size in the *sender* by starting a packet with the <STX> character instead of <SOH>. Like other backward-compatible XMODEM extensions, it was intended that a -1K transfer could be started with any implementation of XMODEM on the other end, backing off features as required.

XMODEM-1K was actually one of the many improvements to XMODEM introduced by Chuck Forsberg in his YMODEM protocol. Forsberg suggested that the various improvements were optional, expecting software authors to implement as many of them as possible. Instead they generally implemented the bare minimum, leading to a profusion of semi-compatible implementations, and eventually, the splitting out of the name "YMODEM" into "XMODEM-1K" and a variety of YMODEMs. Thus XMODEM-1K actually post-dates YMODEM, but remained fairly common anyway.

A backwards compatible extensions of XMODEM with 32k and 64k block lengths was created by Adontec for better performance on high-speed error free connections like ISDN or TCP/IP networks.

Pre-acknowledge

Over reliable (error-free) connections, the receiver could eliminate the latency issue by "pre-acknowledging" the packets. The receiver would already send ACK while the packet was still being transmitted. This effectively breaks error-correction since a packet is always acknowledged regardless of its integrity (which can only be checked after it has been completely received).

Since this feature is only an alteration of the receiver-side behaviour, it does not require any changes in the protocol or on the sender's side.

Pre-acknowledge was also possible for YMODEM. It was made obsolete by variants such as YMODEM-g or ZMODEM.

References

1. ^ Christensen, Ward (1 January 1982). "XMODEM Protocol Overview" (http://www.techheap.com/communication/modems/xmodem.html) . http://www.techheap.com/communication/modems/xmodem.html.
2. ^ Forsberg, Chuck (11 September 1986). "XMODEM/YMODEM PROTOCOL REFERENCE" (http://www.techheap.com/communication/modems/xmodem-ymodem_reference.html) . http://www.techheap.com/communication/modems/xmodem-ymodem_reference.html.

- XMODEM/YMODEM Protocol Reference (http://www.textfiles.com/programming/ymodem.txt) , Chuck Forsberg
- The Adontec XMODEM/32k and XMODEM/64k extensions (http://www.adontec.com/xmodem_e.htm)

Categories: BBS file transfer protocols | 1977 introductions
