How much of PPP was HDLC, an IBM initiative way back in the 1970s and still widely used

The Point-to-Point Protocol (PPP) borrowed **virtually its entire framing structure and error-checking mechanism** from High-Level Data Link Control (HDLC).

When the Internet Engineering Task Force (IETF) designed PPP in the late 1980s and early 1990s, they did not reinvent the wheel for data containment. Instead, they officially adopted **"HDLC-like framing"** (standardised in RFC 1662 in 1994).

The technical breakdown of how much of HDLC is inside PPP, and where they diverge, is outlined below.

The 85% Same: HDLC Enclosure

If you look at a standard PPP data frame, it uses the exact structural blueprint defined by the ISO 3309 standard for HDLC:

- **The Flags (1 byte):** PPP uses the classic HDLC byte `01111110` (`0x7E`) to mark the precise beginning and end of every single frame.
- **The Address Field (1 byte):** HDLC was designed to support multi-drop networks, so it needed destination addresses. Because PPP is strictly a 1-to-1 direct wire (point-to-point), this field is practically useless. To maintain HDLC compliance, PPP simply hardcodes it to the broadcast address `11111111` (`0xFF`).
- **The Control Field (1 byte):** PPP inherits the HDLC Unnumbered Information (UI) frame format, hardcoding this byte to `00000011` (`0x03`) to signal connectionless data tracking.
- **Frame Check Sequence (FCS) (2 or 4 bytes):** PPP directly uses the HDLC standard Cyclic Redundancy Check (CRC-16 or CRC-32) to evaluate whether data bits were corrupted during transmission.

Because the Address and Control bytes are always static constants in PPP (`0xFF03`), network engineers frequently use PPP's Link Control Protocol to compress and strip them out entirely to save bandwidth.

The Critical 15%: What PPP Added

While IBM's 1970s SDLC (which ISO turned into HDLC) was a bare-bones framework meant to push data between predictable mainframes, it lacked the flexibility needed for the emerging public internet. PPP modified or appended three critical features:

Feature Layer	IBM / ISO HDLC	IETF PPP Enhancement
Protocol Multiplexing	Designed to carry only one network protocol per link.	Added a **Protocol Field** (1 or 2 bytes) inside the frame, allowing a single wire to carry IPv4, IPv6, AppleTalk, or IPX simultaneously.

Feature Layer	IBM / ISO HDLC	IETF PPP Enhancement
Link Negotiation	Hardcoded parameters; lines had to be configured identically manually.	Added **LCP (Link Control Protocol)** to let two devices automatically negotiate frame sizes, compression, and loop detection.
Security & Auditing	Assumed a physically secure, dedicated environment; no native login mechanism.	Added native authentication hooks like **PAP** and **CHAP** to securely verify who is on the other end of the wire.

Where It Is Still Used Today

Even though dial-up modems (*where it was used*) have faded away, this exact hybrid architecture remains heavily active under the hood of modern infrastructure.

When you connect to fibre or broadband, it frequently establishes your session via **PPPoE (PPP over Ethernet)**. When you launch specific enterprise or legacy VPN tunnels, they rely on **L2TP (Layer 2 Tunneling Protocol)**, which natively wraps its payloads inside PPP frames. Whenever PPP is invoked, the foundational bit/byte boundaries set by IBM and the ISO more than 50 years ago are still processing your data.

What year was HDLC launched by IBM. How widely was it deployed by 1985 when the IETF looked at it.

IBM officially launched the foundation of HDLC in **1975** under the name **SDLC** (Synchronous Data Link Control), alongside its seminal Systems Network Architecture (SNA).

IBM submitted SDLC to international standards bodies in the mid-1970s. The International Organization for Standardization (ISO) modified the protocol—most notably adding peer-to-peer capabilities—and officially ratified it as **HDLC in 1979**.

By **1985**, when the architectural precursors to the IETF were looking to standardize WAN communication, HDLC and its direct derivatives were **ubiquitous and globally dominant**. It had achieved near-total adoption across three major pillars of the telecommunications and computing industries:

1. The Global Public Data Network Standard (X.25)

By 1985, the primary way computers spoke to each other over long distances across different manufacturers was **X.25**, the dominant packet-switching standard used by telecom monopolies worldwide (such as Tymnet in the US or Datapac in Canada). X.25 used a customized version of HDLC known as **LAPB (Link Access Procedure, Balanced)** as its data-link layer. This meant almost all public data transit already relied on HDLC framing hardware.

2. Mainframe Dominance (IBM SNA)

IBM absolutely dominated corporate enterprise computing in 1985. Virtually every bank, airline reservation system, and insurance conglomerate ran on IBM mainframes. These multi-drop corporate networks used the original IBM implementation of HDLC (SDLC) to connect hundreds of thousands of dumb terminals back to central computing hubs.

3. Chipset and Hardware Mass-Production

Because HDLC framing was used by both IBM and global telecoms, silicon manufacturers like **Intel, Motorola, and Zilog** mass-produced specialized, inexpensive communication controller chips (such as the Intel 8044 or Zilog SCC) with HDLC framing logic baked directly into the silicon.

The IETF's Logic

When the IETF began drafting the Point-to-Point Protocol (PPP) in the late 1980s, choosing HDLC was a matter of **high utility and zero friction**.

- Routers and servers did not need new, custom network cards to run PPP.
- The existing serial line hardware already knew how to calculate HDLC checksums and read HDLC flags at a hardware level.
- PPP simply repurposed an overwhelmingly successful, universally deployed 1970s hardware wrapper to route the brand-new traffic of the open Internet.

Would you like to see how the ISO altered IBM's strict "master-slave" architecture to turn it into the peer-to-peer system that the Internet required?

What part of HDLC meant adding peer-to-peer capabilities from a coding perspective.

From a coding, architectural, and state-machine perspective, adding peer-to-peer capabilities to HDLC boiled down to replacing a rigid **Primary/Secondary** design with a **Combined Station** model running **Asynchronous Balanced Mode (ABM)**.

In IBM's original 1970s SDLC, a central mainframe (the Primary) issued "commands," and terminal controllers (the Secondaries) could only issue "responses" when explicitly spoken to. True peer-to-peer communication meant that both sides of a wire had to be coded identically.

The software logic shifted to accommodate peer-to-peer capabilities through the following changes:

1. Dual State-Machine Merging (The "Combined Station")

In software or firmware, a standard master-slave setup requires two completely distinct software blocks: a command-issuing engine and a passive listener engine.

To achieve peer-to-peer functionality, the ISO introduced the **Combined Station**. The code running on device A and device B became identical. From a structural coding perspective, the driver now instantiated a single logical state machine that multiplexed both behaviors simultaneously:

- **The Transmit Logic** can autonomously send commands without waiting for an invitation.
- **The Receive Logic** handles incoming commands from the opposite side as a peer, processing them concurrently.

2. The P/F (Poll/Final) Bit Bitmask Logic

Every HDLC frame features a 1-bit subfield in its Control byte called the **P/F bit**. In a master-slave configuration, this bit serves a single purpose: the Master sets it to **1 (Poll)** to compel the secondary to talk; the slave sets it to **1 (Final)** to signal it is finished talking.

In a peer-to-peer ABM configuration, the parsing code evaluates the exact same bit entirely differently based on context:

- If the incoming frame is parsed as a **Command**, the software treats a `1` as a **Poll**, meaning the remote peer expects an immediate acknowledgement.
- If the incoming frame is parsed as a **Response**, the software treats a `1` as a **Final**, matching it to an outstanding request the device previously sent out.

3. New Link-Establishment Opcode Control Flags

To transition a link into peer-to-peer mode at startup, developers had to code and handle specific initialization commands inside HDLC **Unnumbered Frames (U-frames)**.

The primary addition was the **SABM (Set Asynchronous Balanced Mode)** command, alongside its later 1980s extension, **SABME** (for extended sequence numbers).

```text
 [Device A]                                      [Device B]


     |                                              |
     | ---- SABM Command (Let's be peers) -------> | (Parser reads U-frame opcode)
     |                                              |
     | <--- UA Response (Unnumbered Ack) --------- | (Acknowledges peer mode active)
     |                                              |
 [Link is now in a balanced, peer-to-peer state]
```

Use code with caution.

When a device boots up, its driver sends a U-frame with the Control byte bit-pattern for SABM (`0x2F` or `0x6F` depending on context). When the receiving peer's parser decodes this pattern, its internal state transition switches from `LINK_DOWN` directly to `BALANCED_PEER_MODE`.

Why PPP Stripped It Back Down

This peer-to-peer ABM logic became so highly optimized that when the IETF built PPP on top of it, they realized they could simplify the coding layer even further. Because Internet routers always treat each other as absolute equals over a serial link, PPP permanently forces the hardware interface into an Unnumbered Information configuration. It handles all its peer-to-peer negotiation through its own sub-protocol (LCP), rather than relying on the older, complex HDLC SABM frame states.

** End of Page