The Technical Development of Internet Email

Craig Partridge

BBN Technologies

Abstract

The development and evolution of the technologies and standards for Internet email took over twenty years, and arguably is still underway today. The protocols to move email between systems and the rules for formatting messages have evolved, and have each been largely replaced at least once. This paper is the story of that evolution, with a focus on why things look the way they do today.

Keywords: Electronic mail, Internet, multimedia communication, protocols

1 Introduction

The explosive development of networked electronic mail (email) has been one of the major technical and sociological developments of the past forty years. A number of authors have already looked at the development of email from various perspectives.[1] The goal of this paper is to explore a perspective that, surprisingly, has not been thoroughly examined: namely, how the details of the technology that implements email in the Internet have evolved.

This is a detailed history of email's plumbing. One might imagine, therefore, that it is only of interest to a plumber. It turns out, however, that much of how email has evolved has depended on seemingly obscure decisions. Writing this paper has been a reminder of how little decisions have big consequences and I have sought to highlight those decisions in the narrative.

2 The Architecture of Email

As the story of this paper is how email came to look the way it does today, we start by describing (in broad strokes) today's world, so that the steps in the evolution can be marked more clearly.

Today's email system can be divided into two distinct subsystems. One subsystem, the message handling system (MHS), is responsible for moving email messages from sending users to receiving users, and is built on a set of servers called message transfer agents (MTAs). The other subsystem, which we will call the user agent (UA), works with the user to receive, manage (e.g. delete, archive, or print), and create email messages and interacts with the MHS to cause messages to be delivered. Readers may recognize this terminology as being roughly that developed by the X.400 email standardization process.

Each subsystem internally has a rich set of protocols and services to perform its job. For instance, the UA typically includes network protocols to manage mailboxes kept on remote storage at a user's Internet Service Provider or place of work. The MHS includes protocols to reliably move email messages from one MTA to another, and to determine how to route a message through the MTAs to its recipient(s).

The UA and MHS must also have some standards in common. In particular, they need to agree on the format of email messages and the format of the meta-data (the so-called *envelope*) that accompanies each message on its path through the network.

The focus of this paper is how these different pieces incrementally came into being and exploring why each one emerged and how its emergence affected the larger email system. In the interests of space, this survey stops around the end of 1991. That termination date leaves out at least four stories: (1) the development of graphics based user interfaces for personal computers and the incorporation of those interfaces into web browsers; (2) the rise of UA protocols such as the Post Office Protocol (POP)[2] and IMAP[3] (these protocols existed prior to 1991, but much of their evolution occurred later); (3) the continuing efforts to further internationalize email (e.g., allowing non-ASCI characters in email addresses); and (4) the rise of unwanted email (dubbed "spam") and tools that sought to diminish it. Furthermore, in the interests of space, we do not consider the development of technical standards for the support of email lists.

3 First Steps

Electronic mail existed before networks did. In the 1960s, time-shared operating systems developed local email systems delivering mail between users on a single system.[4] The importance of this work is that email requires a certain amount of local infrastructure. There needs to be a place to put each user's email. There needs to be a way for a user to discover that he or she has new email. By the early 1970s, many operating systems had these facilities.

In July of 1971, Dick Watson of SRI International published an Internet Request for Comments[5] (RFC-196) describing what he called "A Mail Box Protocol," The idea was to provide a mechanism where the new Network Information Center (NIC) could distributed documents to sites on the ARPANET. Watson described a way to send files (documents) to a teletype printer, with different mailboxes for different types of printers. Mailbox 0 was a teletype

> "assumed to have a print line 72 characters wide, and a page of 66 lines. The new line convention will be carriage return (X'0D') followed by line feed (X'0A')… The standard printer will accept form feed (X'0C') as meaning move paper to the top of a new page."[6]

Ray Tomlinson of Bolt Beranek and Newman (now BBN Technologies or BBN) read Watson's memo and reacted that "it was overly complicated because it tried to deal with printing ink on paper with a line printer and delivered the paper to numbered mailboxes."[7] In Tomlinson's view, the correct approach was to send documents to a user's electronic mailbox and let the user decide if the document merited printing.[8] So Tomlinson set out to see if he could send email this way between two TENEX systems[9] over the ARPANET. His approach was simple.

TENEX already had an existing local email program called SNDMSG[10], which, given a message, appended that message to a file called MAILBOX in a user's directory. TENEX also had a homegrown file transfer service called CPYNET (written by Tomlinson). In a passive mode, CPYNET listened at a particular address for requests to read, write or append to a particular local file. Email was achieved by incorporating CPYNET into SNDMSG. If SNDMSG was given a message addressed to user at a remote host, it opened a CPYNET connection to the remote host and instructed CPYNET to append the message to the user's mailbox on that host.

Users learned that they had received network email the same way they learned they had received local email. In TENEX, they got a "You have mail" message when they logged in. Mail was read by viewing or printing the mailbox file, usually with the TYPE command. (Almost immediately, TYPE MAILBOX was replaced with a TENEX macro READMAIL). Messages were deleted by deleting the relevant lines with a text editor.

Tomlinson made two important contributions. First, he found a way to express the networked email address. He chose to use the "@" sign to divide the user's account name from the name of the host where the account resided, resulting in the now ubiquitous *user@remote* format.[11] Second, SNDMSG was the first MTA – it took a message and delivered it (using the CPYNET protocol) to a remote user's mailbox.

Observe that the last contribution is a surprise. We might imagine that the first program was more of a user agent (UA) rather than a message transfer agent (MTA). But SNDMSG could only deliver mail, it could not receive mail, and it delivered the email all the way to the recipient's mailbox. Therefore, SNDMSG was much closer in spirit to an MTA (and, indeed, as we shall see, was used as an MTA for a number of years). At the same time, SNDMSG was primitive. If there were multiple email recipients on the same host, it copied the message once for each recipient. If the remote host was down, SNDMSG simply returned a failure message – it made no effort to retransmit.

Despite its primitive nature, Tomlinson's creation took off. The next few years saw it mature from a fun idea to a central feature of the ARPANET (and later the Internet).

4 From Primitive to Production

By late 1973, email was widely used on the ARPANET. What happened after Tomlinson's experiment to make this happen? Obviously email met a need. But there were also technical steps: standardization of the transfer protocol and the development of user interfaces.

4.1 A Standard Transfer Protocol

First, the community replaced CPYNET with a standardized file transfer service, the first generation of the File Transfer Protocol (FTP). This process took a while. In 1971, FTP was simply a set of rather complex ideas written up in a set of RFCs by a team led by Abhay Bhushan of MIT.[12] The goal of these ideas was to create a general tool to manage files (including deleting and renaming files) on remote machines and to do it in a way that met the needs of any envisioned application.[13]

At the same time, Dick Watson's mailbox idea was continuing to mature. In November of 1971, a team including Watson proposed a way to enhance (the still nascent) FTP with an explicit "MAIL" command to support appending a file to a mailbox. They further proposed that email be simply ASCII strings of text (no binary images) and that mailbox numbers be replaced with text user identifiers. The identifiers were "NIC handles." NIC handles were given out by the Network Information Center to authorized network users (and were used as login IDs on ARPANET terminal servers, called TIPS). This idea, of course, meant that every host would need to maintain a table mapping NIC handles of local users to the location of their mailbox file. Retaining Watson's original idea of accessing printer, the MAIL command could be given the name "PRINTER" instead of a NIC handle and the file would be printed.

Concurrently, Tomlinson distributed SNDMSG to other TENEX systems and people began to get hands-on experience with email. TENEX was the most common operating system on the ARPANET at the time and so probably at least half the ARPANET users had access to SNDMSG.

In April of 1972, most of the interested parties, including both Tomlinson and Watson, met at MIT to discuss revisions to the File Transfer Protocol. The meeting made several decisions at least one of which proved to have a long-term impact: they agreed to use text (ASCII) commands and replies (previous versions of FTP had used binary commands) to aid interactive use[14]. To this day, the Internet uses text commands to transfer email (and the tradition lives on in much later protocols, such as the web's transfer protocol, HTTP). A new version of the FTP specification, based on these ideas and written by Bhushan, came out in July.[15]

The new specification envisioned that email would be delivered via the APPEND command, which appended data to a file. Discussions about FTP and email continued however, and a month later, Bhushan issued a revision to the FTP specification[16] to include a new command, MLFL (Mail File). It is said Bhushan came up with MLFL because, one evening while he was writing the revision, a fellow graduate student at MIT stopped by to suggest a better solution was required for email.[17]

MLFL took one argument, a user id, which could either be a NIC handle or a local user name (local to the remote host). The user id could also be left out, in which case the mail was to be delivered to a printer. After the MLFL command was accepted, the email file was transmitted over an FTP data channel (with the end of the file indicating the end of message). The file was required to be in ASCII. A separate copy of the file was sent for each recipient at a host.

MLFL was an important step. A key flaw in Tomlinson's prototype email was that you had to know where in the receiving host's file system a user's mailbox was located, so that you could append to it.[18] This limitation probably explains why most of the email activity in 1971 and 1972 appears to have taken place between TENEX systems, where the file name for the mailbox was consistent. MLFL adopted Watson's notion that mailboxes are symbolic names that the receiving system translates into an appropriate user mailbox file and thereby freed email from system specific limitations.

An interactive MAIL command was also defined, so that users logged into a TIP could type in an email message using only FTP's control connection. In this case, a line with a single dot (".") on it marked the end of the message. Ending a message with a single dot is still how email is moved over the Internet today.

The MAIL and, more important, MLFL commands remained the way email was delivered between systems for several years.

In the fall of 1972 Bob Clements of BBN updated SNDMSG to use the new commands. Several other email cognizant FTP implementations appeared. The most notable is probably the system for MIT's Multics. Ken Pogran wrote the FTP implementation and Mike Padlipsky wrote the NETML program that handled email. [19] Multics was exceptional for the time because it had good security including user file privileges, so Padlipsky had to invent a special user (ANONYMOUS) to receive email and distribute it to users.[20] The concept of an anonymous login account caught on as a way to permit FTP access to users who did not have an account and remains a central feature of FTP to this day.

4.2 First User Agents

The second development of 1972 and 1973 was the creation of tools to create and manage email. Here the center of innovation was within the Advanced Research Projects Agency (ARPA) itself. Larry Roberts, head of the ARPA office funding ARPANET, was an early and aggressive user of email. Early in 1972, Stephen Lukasik, the head of ARPA, also began using email and that induced a number of others, including the ARPA department heads, to use email too.[21]

Soon Lukasik became frustrated with READMAIL, which forced him read through all the messages in his mailbox in order. Lukasik liked to keep copies of email he received, which made the problem worse. He appealed to Roberts for something better.

One night in July, Roberts wrote a tool using macros for the TECO (Text Editor and COrrector[22]) text editor to manage a mailbox.[23] The tool was dubbed RD. RD made it possible to list the messages in the mailbox, to pick which message to read next, and to print individual messages.

Roberts' colleague at ARPA, Barry Wessler, promptly rewrote RD as a standalone program in the programming language SAIL and added additional features for usability. Improvements in Wessler's "New RD" or NRD included the ability to manage more than one file of messages, and mechanisms to file, retrieve, and delete messages. RD and NRD were the first mailbox management tools, the first true User Agents.

Wessler's NRD was not distributed outside ARPA. (RD was.) In early 1973, Martin Yonke was a graduate student intern at the University of Southern California's Information Sciences Institute (ISI) and looking for something to do. Steve Crocker of ARPA gave Yonke a copy of Wessler's code (which ran on TENEX) and suggested Yonke look at improving it. Yonke added command completion (type the first letter or two of a command and the rest of the name would be filled in) and a help interface. A user could type a question mark in most places in a command to learn what the choices were. The revised NRD was dubbed BANANARD.[24] (At the time, "banana" was technical slang for "cool" or "better"). Yonke distributed and maintained BANANARD for a bit less than a year though it remained in use for several years more.

There is at least one fun story from that year. BANANARD kept an index of messages in a file so Yonke had to estimate how big the index (which was read into memory) might be. Yonke estimated the largest possible mailbox size, doubled that, and concluded that assuming a mailbox was never larger than 5,000 messages was safe. Steve Crocker exceeded the limit within a few months. So did John Vittal.[25]

One of the challenges in RD and NRD was that there was still no standard format for email messages. Headers varied. It was hard to find where one message ended and the next one started. Wessler remembers trying to get NRD to find the start of headers but it was too hard because messages routinely had other messages embedded in them. Therefore, NRD (and RD and BANANARD) relied on the receiving system to place a start-of-message delimiter before each message in the mailbox.[26] The delimiter had four SOH (Start Of Header, also known as Control-A) bytes followed by information about the message (initially just a byte count, later somewhat more information).[27] In one of those odd quirks, part of the start-of-message delimiter has lived on. While some present-day email systems parse for a header, others still expect messages separated by a line with four consecutive SOH bytes.

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.