The Arpanet IMP Program: Retrospective and Resurrection

David Walden and the "IMP Software Guys"

This article first sketches the history of the Arpanet IMP program, originally developed in 1969, and enumerates a sequence of other systems evolving from the original program over 30 years. The article then describes the redigitization and reassembly in 2013 of the program from a 1973 line printer listing, after which the program was run once again on an emulation of the 1969 computer hardware.

People from Bolt Beranek and Newman (BBN) and others have documented extensively the overall Arpanet technology, including the Arpanet Interface Message Processor (IMP) and its algorithms.[1] However, previous writings have not focused on the decades-long evolving history of the IMP program, as this article does. Then in 2013, years after the program's effective demise, a faded 1973 line printer listing of the IMP program was run through a special optical character recognition (OCR) program optimized to process such historical artifacts; an assembler was recreated to assemble the IMP code (looking like the modified PDP-1 Midas assembler used in 1973); and a software emulator of the original modified Honeywell 516 IMP hardware platform was created. Through these steps, we were able to run the IMP program again, a bit of historical revival we also describe in this article.

Arpanet IMP Program Retrospective

During its first 30 years, the original Arpanet IMP code was reused, reimplemented, extended, and changed. The original Arpanet IMP functions also significantly impacted numerous more or less closely related networks and, to some extent, the Internet today.

Preparation, Implementation, and Installation

In 1968, BBN was preparing to bid on the "request for quotation" from ARPA to develop the Arpanet IMPs.[2] Frank Heart, Bob Kahn, Severo Ornstein, and David Walden were the main members of BBN's proposal team (although other BBN people participated), with Heart as the team leader, Kahn with prior background in the concepts of packet switching, Ornstein as the hardware designer, and Walden as the software designer. Shortly before the due date for BBN's proposal[3] to ARPA, Will Crowther was added to the team as another (more senior) software person. After BBN was awarded the contract, Ben Barker was added as a hardware designer and Bernie Cosell as a third software person. BBN was awarded the IMP development contract with a start date of 1 January 1969.[4]

During the first eight months of 1969, Cosell, Crowther, and Walden developed the IMP's program, and Barker and Ornstein developed BBN's modifications to the Honeywell 516 computer to adapt it for the Arpanet IMP function. Heart and Kahn interacted in various useful ways with the hands-on developers. On the software side of things, Cosell focused on the development tools and IMP code that allowed debugging and statistics taking, Crowther focused on the code that handled interactions among the IMPs, and Walden focused on the code for handling an IMP's connection to host computers. (Nonetheless, all three knew the entire software system inside out.[5,6]) "Host" was the name for computers connected to an IMP and using the network of IMPs to communicate with others host computers.[7] The Host/IMP interface was specified by BBN and was known as the "1822 interface."[8]

The basic Honeywell 516 computer,[9–11] on which the IMP system was based, had 32 thousand bytes of RAM and a 1-microsecond cycle time (1 MHz).[12] It was also the size of a refrigerator. The BBN-designed modifications included interfaces from the 516 to inter-IMP communication circuits and to the host computers using the Arpanet, as well as a few

 Published by the IEEE Computer Society 1058-6180/14/$31.00

other changes to simplify interrupt programming and help with system reliability.[13]

However, the IMP software was not written and assembled on the 516. Rather, the IMP program was composed, edited, and assembled on BBN's PDP-1d computer. The assembled program in octal was then output on paper tape for loading into the IMP, via its paper tape reader, for debugging. After BBN's PDP-1d was connected to the Arpanet in 1971, new IMP software could be loaded into IMPs via the network itself.

Doing software development on the PDP-1d computer (rather than on the 516 standing alone in the development lab) allowed the following:

- IMP programmers could do assembly work simultaneously because the PDP-1d was a time-shared computer.
- IMP programmers could use the PDP-1's powerful TECO editor[14] for program composition and editing.
- IMP programmers could use the powerful macroprocessor of the PDP-1 Midas assembler[15,16] to define the instruction codes for the BBN modifications to the 516 (as well as for defining the basic 516 instruction codes and 516 memory page size to Midas) and to add other aids for developing an optimized and reliable software system.[17]
- The edit-assemble cycle could be repeated until assembly errors were gone without carrying paper tapes to the 516 additional times.
- Assembly listings could be printed on the line printer connected to the PDP-1d.[18]

Starting around Labor Day in 1969, one IMP a month was delivered in succession to the University of California, Los Angeles (UCLA), the Stanford Research Institute (SRI), the University of California, Santa Barbara (UCSB), and the University of Utah. Early in 1970, a fifth IMP was installed at BBN (see Figure 1), and a network control center capability was developed by late 1971.[19,20]

The Arpanet, which became operational as a research network in early 1970, directly transitioned to or somewhat influenced a number of other networks.

Evolution of the Arpanet and Its Successors and Derivatives

Between 1969 and 1975, the Arpanet expanded at the rate of about 10 IMPs per year (to approximately 60 IMPs total). Between

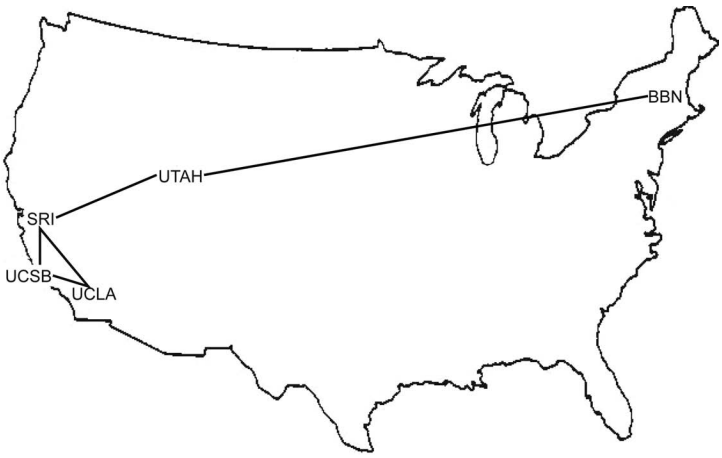

Figure 1. Early 1970 Arpanet map. By that time, the Arpanet spanned locations in Los Angeles (UCLA); Menlo Park, California (SRI); Santa Barbara, California (UCSB); Salt Lake City, Utah (University of Utah); and Cambridge, Massachusetts (BBN).

1 July and 31 December 1975, the Arpanet operation was phased over from ARPA to the Defense Communications Agency (DCA) to be run as an operational network, although it continued to be involved in communications R&D experiments. By 1982, there were about 90 IMPs in the Arpanet and the government decided that the Arpanet would be the foundation of the Defense Data Network (DDN). The creation of DDN involved splitting the existing Arpanet into two parts—one that was still known as Arpanet and served nonmilitary users, and one known as Milnet and served military users. (The two parts could communicate in a controlled way through *mail bridges*.) Also over this period, the Arpanet (in its pre-DDN and DDN forms) became a part of many early Internet experiments and thus the backbone of the Internet in its early growth period. In 1988, steps began to dismantle the Arpanet.[21]

From relatively early in its Arpanet effort, BBN also had aspirations for expanding upon and exploiting the Arpanet technology beyond the Arpanet, both elsewhere in the government and commercially. Relatively early on, a slightly modified version of the 516 IMP technology was deployed in the US intelligence community.[22] In 1972, BBN organized a commercial packet-based telecommunications carrier known as Telenet, which originally used a version of the 516 IMP code Telenet's programmers (not the BBN IMP guys) converted to run on a different Honeywell X16-series computer,[23] but Telenet later built its own computer and wrote its own software for that. BBN made consulting

Many of the technology approaches originally demonstrated in the Arpanet have become part of the DNA of modern data communications (such as the Internet).

agreements with Logica in the UK and SESA in France for them to bid on the 516/316 IMP technology (we describe the 316 technology later), and BBN bid on other commercial networks (based on Arpanet technology) itself, for instance at Citibank and On-line Systems. By 1974, BBN had developed the Pluribus parallel processor for which it developed software interoperable with the 516/316 IMP system. In time, BBN started its own computer company (BBN Computer Corporation, BBNCC), which originally handled the hardware maintenance contracts for BBN-delivered networks (as well as trying to be a minicomputer vendor more generally). By 1982, BBNCC changed its name to BBN Communications Corporation and refocused its activities on networking. From this part of the company, BBN delivered many commercial, military, and other networks around the world (see http://walden-family.com/impcode/#networklist). BBNCC supported the 516/316 IMP technology and the Pluribus IMP technology, and it developed new IMP technology based on BBN's C/30 and C/300 computers. A partial overview chart of these various systems is available at http://walden-family.com/impcode/hwchart.pdf.

BBN's Arpanet technology also influenced the design of several non-BBN networks. The LFK Network was a deliberate copy of the Arpanet code but with the IMP algorithms rewritten from scratch to and run on Norsk Data computers.[24] The founders of the Packet Communications Corporation (who left BBN a little before Telenet was founded) had a copy of the IMP system listing (BBN was required to give copies to companies that requested it for a slight handling charge) but perhaps did not closely follow BBN's IMP system design. Other places—for example, at least one company in Japan—studied the function of the original Arpanet IMP, and it influenced the design of their network.

More generally, many of the technology approaches originally demonstrated in the Arpanet have become part of the DNA of modern data communications (such as the Internet): dynamic routing, operation without central control, packet flow control and reliable transmission, its operating philosophy, its engineering approach to standards, and so on.

Evolution of the IMP Code

A main point of this article is to trace the evolution of the original software developed for the 516 IMP as it was reused and adapted for several different hardware platforms over many years. A partial overview chart of this evolution resides at http://walden-family.com/impcode/swchart.pdf.

The 516/316 and Network Algorithms. An early step (1971) was reusing the 516 IMP code on a Honeywell-316-based computer. The 316 computer had a faster cycle time than a 516, was not as ruggedized as the 516 but was still in a refrigerator-sized box, and had other electronic technology changes. The BBN-designed modifications to the 516 to turn it into an IMP also had to be made to work for the 316. However, these hardware changes did not require a different IMP program than what ran on the 516. The first application of the 316 IMP was as part of the Arpanet Terminal IMP (TIP[25]). The TIP was an option for embedding within the physical IMP box a capability for 63 terminals to connect directly to an IMP independent of a normal host computer. This was accomplished with additional BBN-developed software: (a) the additional TIP software resided in 316 memory beside the IMP code; (b) it handled BBN-developed "terminal concentrator" hardware to which local and dial-up terminals could connect; and (c), from the viewpoint of the IMP software, the "TIP host" looked pretty much like other host computers. Later, 316 IMPs were installed without the TIP option where ruggedized 516s were not needed. Thus, from 1971 on, this IMP system software was thought of as being for the 516/316 IMP.

From the time of the initial IMP installations in 1969, the Arpanet continued to expand. As it became a more or less operational network, fixes were required and improvements could be made to the 516/316 IMP software. Later the algorithms were reimplemented so Pluribus-platform-based IMPs could function compatibly with the 516/316 IMPs, and the 516/316 IMP program was adapted to run on the C/30 platform. However, it was years later before fundamental networking algorithms had their initial development on a platform other than the 516/316 IMP platform.

We will sketch the initial round of changes to the IMP software by quoting from a 1972 paper:[26]

> A balanced design for a communication system should provide quick delivery of short interactive messages and high bandwidth for long files of data. The IMP program was designed to perform well under these bimodal traffic conditions. The experience of the first two and one half years of the ARPA Network's operation indicated that the performance goal of low delay had been achieved. The lightly loaded network delivered short messages over several hops in about one-tenth of a second. Moreover, even under heavy load, the delay was almost always less than one-half second. The network also provided good throughput rates for long messages at light and moderate traffic levels. However, the throughput of the network degraded significantly under heavy loads, so that the goal of high bandwidth had not been completely realized.
>
> We isolated a problem in the initial network design which led to degradation under heavy loads.[27,28] This problem involves messages arriving at a destination IMP at a rate faster than they can be delivered to the destination Host. We call this reassembly congestion. Reassembly congestion leads to a condition we call reassembly lockup in which the destination IMP is incapable of passing any traffic to its Hosts. Our algorithm to prevent reassembly congestion and the related sequence control algorithm are described [later in the quoted paper].
>
> We also found that the IMP and line bandwidth requirements for handling IMP-to-IMP traffic could be substantially reduced. Improvements in this area translate directly into increases in the maximum throughput rate that an IMP can maintain.

Another set of changes was made to expand the capabilities rather than the performance of the IMP:

> The size of the initialization code and the associated tables deserves mention. This was originally quite small. However, as the network has grown and the IMP's capabilities have been expanded, the amount of memory dedicated to initialization has steadily grown. This is mainly due to the fact that the IMPs are no longer identical. An IMP may be required to handle a Very Distant Host [a host at the other end of a communications circuit rather than directly wired to the IMP], or TIP hardware, or five lines and two Hosts, or four Hosts and three lines, or a very high speed line, or, in the near future, a satellite link. As the physical permutations of the IMP have continued to increase, we have clung to the idea that the program should be identical in all IMPs, allowing an IMP to reload its program from a neighboring IMP and providing other considerable advantages. However, maintaining only one version of the program means that the program must rebuild itself during initialization to be the proper program to handle the particular physical configuration of the IMP. Furthermore, it must be able to turn itself back into its nominal form when it is reloaded into a neighbor. All of this takes tables and code. Unfortunately, we did not foresee the proliferation of IMP configurations which has taken place; therefore, we cannot conveniently compute the program differences from a simple configuration key. Instead, we must explicitly table the configuration irregularities.

John McQuillan has also said the following about that era, during which checksums and other code robustness devices were put into the code:[29]

> [A] significant part of the effort I put in to the IMP program from 1971 to 1973 had to do with hardware/software interactions. The interrupt system of the 516, and the direct memory channels turned out to be a key focus, both as strengths of the hardware, and sources of issues and failures … One of the goals in that period was to make the IMP more resilient.

After the noted changes were made, the major effort of the next few years was redoing the original Arpanet routing as the network grew and the limitations of the original algorithm were discovered. First there were small modifications, and then McQuillan looked at the issues in detail.[30] Eventually, McQuillan and others developed a new routing algorithm[31,32] that the IMP programmers implemented. The original routing algorithm was useful for getting the Arpanet up and running quickly and supporting, more or less, its first few years of operational use. The new routing algorithm lives on today in the Open Shortest Path First (OSPF) routing algorithm[33] used throughout the Internet. Although not unique to the routing transition, the routing

transition was an instance in which incompatible releases of the IMP software had to be distributed. This added significant complexity to the release effort (an interim release had to be created to allow moving between the prior and new operational releases).

Pluribus. As the 516/316 IMP software evolved, some of the original 516 IMP development team moved on (and were joined by other engineers and programmers) to develop the Pluribus multiprocessor system. The Pluribus was based on a Lockheed Sue processor but most of the hardware system was developed and fabricated by BBN (which also acquired SUE processor manufacturing). With a different instruction set and a much different computer architecture, the IMP algorithms had to be completely recoded for the Plurbus-based IMP.[34,35] In addition "reliability code" was developed to allow a Pluribus IMP to keep functioning as a packet switch in the face of various bits of its hardware failing, such as a processor or memory.[36,37] This was so successful that there was no simple off switch for the machine; a program had to be run to shut parts of the machine down faster than it could "fix itself" and keep running.

The Pluribus IMPs and 516/316 IMPs ran together compatibly in the Arpanet and later in DDN. In particular, the complex transition to the new Arpanet routing algorithm mentioned previously had to be done in the 516/316 IMP and Pluribus IMP in parallel.

MBB-Based Systems. In 1978, BBN developed its Microprogrammable Building Block (MBB) computer, which users could code to perform different functions.[38,39] The Honeywell 316 computer was going out of production and the Honeywell 716 was not suitable for the IMP task. Thus, microcode was created for the MBB to make it look like a 316 IMP, and by the fall of 1979, the MBB-based system was functioning as an IMP packet switch called the C/30. The 316 IMP code was not changed much for its transition to the MBB emulating the 316 in microcode. (Another version of the MBB was microcoded to more or less directly execute the C programming language in which Unix was written, and these Unix machines, called C/70s, were used in networks of C/30 packet switches as terminal concentrators—like the TIP option for the 316-IMP—and to run network-control-center software.) Almost 1,500 MBB-based systems were installed in networks delivered by BBN. Most of this work was done by BBNCC.

The efforts in the early years of BBNCC primarily consisted of adding capabilities to the edges of a C/30 network. One such effort (circa 1982) was adding an interface for X.25 hosts to the IMP software. The X.25 interface initially ran on top of the standard Arpanet 1822 host interface.[40] This system simultaneously supported the Arpanet 1822 and X.25 interfaces. Around the time the X.25 interface was added, BBNCC began calling the system a packet-switch node, or PSN (more consistent with usage in the networking marketplace). BBNCC also developed new TIP-like terminal concentrators for this line of packet switches, called terminal access controllers or TACs (ca. 1980–1982),[41,42] and packet assembler/disassemblers or PADs (1981–1983), which were terminal concentrators in a computer separate from the packet switch.[43]

During the 1981–1983 period, BBNCC also had to deal with the conversion of the networks it was operating from the original Arpanet host-to-host protocol (the Network Control Protocol, or NCP) to the use of TCP/IP as the host-to-host protocol.[44] BBN had to adapt its own systems functioning as hosts to TCP/IP and was involved in staging the transition for other hosts.

In 1982–1983, the IMP code underwent a major change as new microcode (known as the Native Mode Firmware System, or NMFS) was written for the MBB. This microcode allowed the IMP software to run on a highly improved 316 (with a 20-bit rather than 16-bit address space, scheduling and process-management functions moved into the microcode, external interface drivers that serviced interrupts and moved bytes from the hardware to memory, and some additional instructions for queuing functions).[45]

In 1983–1985, BBNCC built a new hardware machine, the C/300. This was a faster MBB, but with the customizations for the Honeywell emulation built into the hardware rather than being in microcode, so it could not run in C/70 or other hardware modes.[46] This was the workhorse in the DDN upgrade.

Note that many of the described MBB-based developments concerned running the IMP code on a succession of upgraded platforms and adding new capabilities at the periphery of a network, such as handling X.25 hosts and communication with a rewritten network control center program. Naturally, there were also upgrades to the basic packet-switching algorithms. Then, in 1985–1987, there was a major change in the basic packet-switching algorithms: the end-to-end code for

the packet switch was rewritten to embed X.25 as part of the basic system rather than having it ride on top of the Arpanet's 1822 interface. This was the PSN 7 version of the system.

In 1987–1989, another major change to the basic packet-switching algorithms was made: the packet-switching store-and-forward code was redone to include congestion control. This was the PSN 8 version of the system.

BBN continued delivering operations networks for several more years but over time divested itself of its operational network business. The original IMP program from which so much had evolved was a thing of the past.

Between 2002 and 2013, efforts were made to recover a digital copy of the IMP program.

Arpanet IMP Program Resurrection

Between 2002 and 2013, efforts were made to recover a digital copy of the IMP program. To the best of our knowledge, no computer file of the program still existed, but we did have a 1973 line printer listing. A good scan of that old listing posted on the Web would be useful to computing historians, and other possibilities could be imagined if the listing could be digitized to the point where it was computer readable. Here we describe those efforts.

Recovering the 1973 516/316 IMP System Listing

In 2002, Paul Wexelblatt, who had been a member of BBN's Arpanet IMP team at one point, was cleaning out his basement. He asked Bernie Cosell and Dave Walden whether they wanted any of his old BBN reports, manuals, and program listings, which included a September 1973 listing of the 516/316 IMP program, IMP version 3050. It was agreed that Walden would pick up some boxes of materials from Wexelblat's house, keep them for a while as he worked on the BBN computing history book,[47] and then offer Cosell anything he wanted for permanent possession. In particular, Cosell asked that the IMP program listing be passed to him, and eventually it was.

In 2009, Tony Michel (also a BBN Arpanet team member) began to think about running the IMP code again on simulated machines, and Cosell passed the IMP program listing to Michel for scanning and OCR. Over the next couple of years Michel did a lot of scanning work but none completely successfully, and he passed the listing to Walden to continue attempts at scanning.

In 2012, Jack Haverty (who early in his career had worked at BBN, close to the IMP system) began asking questions about the Arpanet IMP software system for a legal dispute about prior art for which he was to be an expert witness. Haverty's idea was that the IMP system represented an example of prior art that could help refute an invention claim from a later date. Considerable discussion went on between Harverty, Cosell, Walden, and a few other ex-BBN IMP people about exactly what the program did. These discussions continued into 2013.

At some point, the idea arose (among Haverty, the law firm, and the firm's other technical experts) of getting a good scan of the listing and OCR'ing it to recover a source code file that could perhaps be assembled and run on simulated IMP machines. Walden still had the listing in hand and offered to take it to the Boston office of the law firm for which Haverty was consulting to get a good quality scan done. (All the scans and listings mentioned in this section and the next are available for study at http://walden-family.com/impcode/.)

However, it was not possible to get decent OCR from the scan. Several OCR programs were tried but none produced good results. (Still, Walden has posted the scan of the listing on his website with other historical IMP system content.)

Hence, Charlie Neuhauser, who was a technical consultant to the law firm, and his colleague Tom Kilbourn hired someone to retype the entire listing, including the octal representation of the assembled program. They then had other people proofread the entire listing—one person reading out loud the newly typed version and the other person checking that against the scan. This caught about 40 errors in the retyped listing, and Charlie caught another 10 (for example, the letter "O" substituted for zero, and the letter "I" for the number "1") while trying to understand the code. The typing job was amazing.

In parallel, James Markevitch became aware of the scan of the IMP system listing on

Walden's website and ran it through his home-built OCR program optimized for faint line printer listings. James explained as follows:[48]

> In about 2006 I was attempting to OCR some old computer listings and found that none of the commercial software packages handled these old listings very well. As a result, I wrote a set of software optimized for processing these old computer listings. I used that software to convert the IMP listing scan into a plain text file.
>
> Old listings have a lot of artifacts that need to be dealt with by the OCR software including broken characters, horizontal and vertical shifting of the characters due to mechanical imperfections in the old printers, light or inconsistent printing due to faded ribbons, and even dot matrix character representations used by some printers, among others. Add to this the fact that the listings are often many decades old (the oldest one I converted was from the late 1950s) and have faded, have been scribbled on, and/or have accumulated stains. Furthermore, the scans are often done at low resolution and many times haven't been fed properly (or are hand-placed) which skews the pages significantly.
>
> The software uses a variety of pattern recognition and geometric techniques to differentiate between characters and create models of expected character positions. Each algorithm is designed to be self-refining and is highly tolerant of noise, allowing the software to extract character images from the artifacts mentioned above.
>
> I also created a Midas assembler[49] and ran it on the processed listing. The output it generates is identical to the listing. This involved reverse engineering the apparent behavior of Midas since it is similar, though not identical, to the PDP-6 and PDP-1 versions. The Midas assembler is written in Perl.[50]
>
> I have done this same sort of thing numerous times: OCR a listing, convert to a file that can be input to an assembler,[51] write an assembler, assemble, compare the output with the OCR'ed listing, and iterate [52] fixing any assembler issues or OCR errors until the OCR output and the assembler output match. Because of the redundant information in assembly listings (both source code and the object code are contained in it), this iteration works very well to quickly catch the occasional OCR errors in all but the comment fields of the listings. Even an OCR error every few pages quickly jumps out with this process. Many old assemblers had undocumented features or behavior and a listing that uses a rich set of the assembler features will expose those behaviors—as a side-benefit, this process results in a usable assembler for this old architecture.

Neuhauser and his team used James's assembly-checked OCR output to check and correct their retyped version of the listing—only another three or so errors. Here is Neuhaurser's story:[53]

> Initially, I went through the entire listing and extracted all the IO opcodes. From this, the Honeywell manuals[9,10] and a review of the functions in the listing, I managed to reconstruct most of the functional aspects of the I/O interface in the sense that I knew there was an OCP or SKS code that did something, but I did not always know what it did. The original ARPA net proposal document[3] was very, very helpful and in my experience was a model of descriptive excellence. From this review I wrote a small document that described the IO codes and what I thought they did.[54] Of course, in some cases it was just rank speculation.
>
> Even without the emulator extension, Tom and I managed to step the emulator through the stand alone debugger and with a few selective modifications to the code (e.g., killing the watchdog timer) managed to get the standalone debugger to work. As a guide to the debugger we used the big three page comment section that preceded the standalone debugger [in the IMP code listing] and found that all the functions we could test (i.e. those that did not use the network) seemed to function as advertised. When looking through the code one thing I found very helpful was the concordance, which was almost as good as using an editor to search the code.
>
> At this point I realized that we would really need some help on the emulator and brought in Robert Armstrong, who had some really good background working with [Digital] PDP [computer] emulation. For a while he even sold a PDP-8 front panel that you could connect to your emulator and key in your code if you wanted to. One thing that Bob really knew is how the simh computer emulator[55] was supposed to be used. By this I mean that he knew that you needed to add commands to the emulator so you could control the state of certain things that would normally be hardware controlled. For example, he made the IMP number adjustable. Also he had commands to configure various devices and to disable them so that we could debug around them.
>
> One thing he did early on was to formalize the extensions he was going to make so that we could all see what the end product was going to be. I thought that this was a very smart thing to do because I was sort of focused on "get it done" and he was focused on "get it right," which was the correct thing to focus on.

We had a number of problems initially. As I see it, the IMP code is not your ordinary garden variety code. In some ways it is more complex than a typical small operating system.

Because timing is so important in the IMP code, the emulator must be more accurate than what we would normally expect for simulating an old computer running its software. In the IMP, as I understand it, you have to do things at the required pace, not too fast and not too slow. This seems to me to have been the source of many of the problems. Bob spent quite a bit of time experimenting to get the emulator timing exactly right. The other thing that impressed me about the IMP code is that it made use of every known trick. Not just self-modifying code for simple things like jump tables, but self-modifying code to control the state of execution. Also the basically interrupt driven nature of the code makes it inherently difficult to deal with. This of course is where emulation really helps because you have a complete visibility into the code that was not available in 1973. Several times Bob wrote simple extensions to the emulator that would allow us to trap on certain conditions that would have been impossible to do even on the real system. For example, he could trap on a particular data write pattern. In the emulation of these old machines the speed advantage of emulation is so great that it is almost like having a logic analyzer attached to the IMP.

One thing that took me some time to understand was what was the 1973 code. Although I assumed that it was actual released code, I always kept in the back of my mind that it might be some sort of test branch from the mainline code and might actually contain bugs. As I worked with the code, and especially when you and the others managed to advance it so far, I became more confident that it was production code.

On a philosophical level it is amazing that this was possible, not just technically, but also from a practical standpoint. Without a doubt the Arpanet was a seminal project. Certainly there were other store and forward systems in use in the same or earlier time frame. I worked on one at Bell Labs. But the Arpanet was the one that was easily scalable (you may not agree given how hard you worked on the code). It was also the one that was universal in the sense that it could connect any sort of device. Of course, I imagine that no one knew what would results when the first machine was turned on. Otherwise, the original code would have been saved. That anything still exists is remarkable. Technology advances so rapidly that even important contributions like the Arpanet are lost after only a few years. Although no one thing is responsible for the Internet as we know it today, Arpanet was certainly important to getting things off the ground and should be remembered for the truly amazing outcome.

IMP Software Cycles Again

As mentioned previously, Bob Armstrong built the simulator for the 516/316 IMP. He started with the existing Honeywell 316 simulator[56] based on the simh simulator,[55] and then he modified the H316 simulator to simulate the instructions and interfaces BBN added to the 316 computer to run the IMP system[11] as part of the IMP system development.[1,5,8] The software kit for Bob's IMP simulator is available on the Web. [57]

Here is Bob's version of the story:[58]

> I originally knew nearly nothing about the IMP or the H316, but I am quite familiar with simh. When Tom and Charlie started using simh, I got involved by answering some of their questions. They were trying to demonstrate the IMP software on simh but were being stymied by the fact that a large chunk of the IMP hardware was custom made by BBN and simh naturally knew nothing about those devices. Eventually the topic of modifying simh came up and I was asked if I thought I could do it.
>
> The then current simh was able to simulate a standard H316 CPU including a number of peripherals—disk drives, magtape, line printers, etc.—none of which were used by the IMP software. On the other hand, the IMP software, although it could run on a standard H316 CPU, required a number of special peripherals none of which simh implemented. Those include:
>
> - A BBN engineered synchronous modem (e.g., Bell 303) interface. This was a fairly sophisticated interface that not only could transfer data directly to and from memory using DMA, but could also frame packets, compute and verify CRCs, do DLE stuffing, and more.
> - A BBN engineered synchronous serial host interface. This is another sophisticated interface that could handshake with the host, detect various host ready and error conditions, convert between the native host word size and the 16 bit H316, and more.
> - A BBN engineered real time clock. Ironically, Honeywell offered a real time clock option for the H316 CPU already, but the IMP used its own version that was similar to, but not compatible with, the Honeywell model.
> - A BBN engineered watch dog timer (WDT). This was a fairly simple device which would fool the CPU into taking a memory protection fault trap if a certain time interval elapsed without the WDT

being reset. Honeywell offered a memory management option for the H316, but this was unused and the corresponding hardware absent on the IMP, and reusing the memory protection fault trap was a simple way of getting a non-maskable interrupt.
- A BBN engineered "task shuffling" interrupt, which was used by the IMP software to implement multitasking.
- A BBN engineered light panel, which contained sixteen status lamps.
- Several BBN added miscellaneous instructions. These included instructions to return an IMP node number, which was hardwired with jumpers in each IMP's hardware and instructions for the MLC [TIP multi-line controller] which was not used in the part of the IMP code we were demonstrating and which we elected not to implement.
- Lastly, the IMP used an individually vectored priority interrupt system. This was actually a standard Honeywell option for the H316 and wasn't engineered by the BBN team, however the existing simh was unable to simulate this option and it had to be added.

This was a considerable number of devices and features that needed to be added to simh, but in principle at least most of them were fairly straight forward to implement. The biggest issue was a lack of concrete information about how any of these devices were actually supposed to work. Charlie read the IMP source code and made a list of all the I/O instructions and made an initial guess as to what they were supposed to do.[54] Later on the BBN IMP guys found and passed along more period documentation from BBN[59,60] that clarified more of it, but there were still a few things we only discovered the hard way, by stepping through execution of the IMP code.

The modem emulation was particularly difficult to get right and I ended up completely rewriting that part at least once. The initial implementation used TCP to tunnel a virtual modem connection between two simh instances, however with TCP network delays are unpredictable and inconsistent, and that proved to be a huge problem. The IMP code is extremely sensitive to modem timing, and even goes to the trouble of measuring, using the real time clock, the exact time it takes to send a message. Knowing the time and the size of the message, the IMP code computes the effective throughput for each modem line, and it needs that number to fall within a very narrow window. If the modem isn't running at the right speed, the IMP will declare the line "down" and attempt to report the problem back to the Network Control Center. This was, as the BBN IMP guys explained to me, because back in the early 1970s an important feature of the IMPs was being able to detect something apparently wrong with inter-IMP lines as early as possible in order to be able to report the problem to AT&T Long Lines. The only solution for the simulator was to rewrite the modem to use UDP, which gives shorter delays and more consistent timing, and to further virtualize the modem simulation and tie it to the simulated real time clock. This makes the modem timing appear completely constant to the IMP code, even if in actuality it is not.

Even with all the IMP specific issues we had, it was also possible that an apparent bug in running the IMP program was the result of a change we'd made to simh. One especially nasty problem, that took a couple of days to track down, turned out to be a very subtle bug in simh's modeling of the TTY interrupt timing. There was a particular combination of Teletype I/O instructions which would not have interrupted on a real H316, but generated an immediate interrupt in simh. This a bug in simh, despite simh already being able to run a number of other generic H316 programs including the official Honeywell diagnostics. Apparently nothing other than the IMP code used this particular combination of Teletype I/O, and the bug was never discovered until we came along.

All told, modifying simh to run the IMP code proved to be quite a bit more challenging than I'd expected. I think I originally promised Charlie that the job would take about two weeks, and in the end it took closer to two months.

By the summer of 2013, with interaction with Neuhauser, Barker, Cosell, Michel, and Walden, Armstrong had refined his understanding of how the IMP hardware and software worked and had the IMP simulator working reliably. Three IMPs in a series could communicate with each other, thus demonstrating that store-and-forwarding of packets and dynamic routing worked; one could use the simulated Teletype of one IMP to inspect and change memory of another IMP, thus indicating that most of the host-handling code worked; IMPs could reload from each other; and so on. The 1973 version of the IMP, from four years after the IMP code originally cycled in 1969, was running again.

On 5 July, the simulator was configured with the following 5-IMP network:

- IMP 1 connected to IMPs 2 and 3
- IMP 2 connected to IMPs 1, 3, and 4
- IMP 3 connected to IMPs 1 and 2
- IMP 4 connected to IMP 2 and 5
- IMP 5 connected to IMP 4

This configuration was representative of the first four- and five-node Arpanet configurations in 1969 and early 1970, as Figure 1 shows: IMP 1 = UCLA, IMP 2 = SRI, IMP 3 = UCSB, IMP 4 = University of Utah, and IMP 5 = BBN. The simulated IMPs were started in numeric order representing the order of their actual installation approximately 43 years ago.

A few days later, a pair of simulated IMPs communicated from different computers using the Internet as a telephone line between them.

As a result of the effort to make IMP version 3050 run again and the consequent decision to write this note, we put out a call for inputs to other members of the Arpanet IMP development and maintenance community. Thus, we now have a 1971 July NCC listing, a 1971 December IMP listing, and a 1974 mid-year listing, which John McQuillan had in his personal archives. John scanned them, and James Markevitch OCR'ed them, making additions to his Midas IMP-code assembler as necessary. These listings also have been posted on the Web.[20,61,62] We also have a scan of a listing (found on the Internet) for a version of the Pluribus IMP (see http://walden-family.com/impcode), and we have listings on microfiche (preserved by Cliff Romash) of the PSN 7 and PSN 8 derivatives of the IMP system.

Conclusion

The first half of this article is a relatively straightforward historical account, albeit an extended anecdote collected from many people rather than a formal piece of computing history research. The second half describes an effort that is part of a relatively new area of computing history work (that some call "living history" and others call "retro history"), in which artifacts from computer history are brought back to life. This latter part of the story might not be finished—we plan to continue the effort to collect, organize, and make such IMP artifacts available to the computing history world.

Acknowledgments

The 1969 BBN Arpanet IMP development team (and the evolving team members over the years) called themselves "the IMP guys," a name that stuck even after women joined the team. Many people have helped write or provided information to this article, and thus we think it is an appropriate homage for the article's author line to include the "IMP Software Guys." We include within this designation the non-BBN people from Silicon Valley who participated in the resurrection of the original 516 IMP program in 2012–2013. Many of these contributors to this article and to the work described herein are listed at http://walden-family.com/impcode/#participants. No doubt some people have been left off that list; we thank them too.

We also thank the anonymous editor and referees for their suggestions for improvement. In addition, Andy Russell and James Cortada read one draft of this article and made suggestions, and Jason Armistead and David Gesswein pointed out an error in another draft.

References and Notes

1. F.E. Heart et al., "The Interface Message Processor for the ARPA Computer Network," *AFIPS Conf. Proc.*, vol. 36, 1970, pp. 551–567; http://walden-family.com/public/1970-imp-afips.pdf.
2. Arpanet Request for Quotations, Defense Supply Service–Washington, Dept. of the Army, 29 July 1968; http://walden-family.com/bbn/arpanet-rfq.pdf.
3. Bolt Beranek and Newman, "Interface Message Processors for the ARPA Computer Network," BBN proposal no. IMP P69-IST-5, 6 Sept. 1968; http://walden-family.com/bbn/arpanet-prop-ocr.pdf.
4. See "Networking at BBN," http://walden-family.com/bbn/#networking, for background information about the BBN IMP development.
5. BBN, *Initial Design for Interface Message Processors for the ARPA Computer Network*, BBN report 1763, 1 Jan. 1969; http://walden-family.com/impcode/1969-initial-IMP-design.pdf.
6. BBN, *Operating Manual for the Interface Message Processors: 516 IMP, 316 IMP, TIP*, BBN report 1877. This manual was revised several times. The April 1973 revision is at http://walden-family.com/impcode/bbn-report-1877.pdf.
7. By this definition, a personal computer connected today to a router or a company webserver connected to a router are host computers, using the internetwork of routers to communicate with other computers.
8. R.E. Kahn, *Interface Message Processor: Specifications for the Interconnection of a Host and an IMP*, BBN report 1822, 1 May 1969. This report was revised a number of times over the years. The January 1976 revision is available at http://

walden-family.com/impcode/BBN1822_Jan1976.pdf.
9. Honeywell, *Series 16 Hardware Documentation*, Apr. 1973; http://walden-family.com/impcode/70130072176C_316_CPU_Descr_Apr73.pdf.
10. Honeywell, *Models 316 and 516 Programmers' Reference Manual*, no. 1970; http://walden-family.com/impcode/70130072156_316_516_PgmrRef_Nov70.pdf.
11. Honeywell, *Intermodem* [mistitled, should be "Interface"] *Processor System Instruction Manual* (Honeywell documentation of the BBN-design-Honeywell-fabricated changes to the 515 computer to make it an Interface Message Processor), Jan. 1970; http://walden-family.com/impcode/imp-hardware.pdf.
12. By way of comparison, a typical laptop computer in 2013 has a RAM size measured in gigabytes and a cycle time measured in gigahertz (although it is hard to make valid cycle and instruction time comparisons given how different computer architectures are now versus then). It is hard for some people to imagine today how an entire packet-switching system could be implemented in a computer as small as the 516 IMP.
13. There is more about the modifications in the "IMP software cycles again" section.
14. D. Murphy, "The Humble Beginnings of TECO," *IEEE Annals of the History of Computer*, vol. 31, no. 4, 2009, pp. 110–115.
15. R.A. Saunders et al., MIT PDP-1 Midas memo, date unknown, http://archive.org/details/bitsavers_mitrlepdp1_1535627.
16. BBN, "Hospital Computer Project," PDP-1 Midas manual, memo no. 6-E: programming software status report, 1 May 1966; http://walden-family.com/impcode/bbn-1966-pdp1d-midas-manual.pdf.
17. B.P. Cosell, J.M. McQuillan, and D.C. Walden, "Techniques for Detecting and Preventing Multiprogramming Bugs," *Minicomputer Software*, J.R. Bell and C.G. Bell, eds., North-Holland Publishing, 1976, pp. 301–306; http://walden-family. com/impcode/detect-interrupt-bugs.pdf.
18. As an aside, the original paper about the IMP from the BBN developers[1] describes editing on the PDP-1 and outputting a paper tape of the symbolic assembly code and assembling that on the Honeywell 516. We did that only a few tedious times before switching to assembly on the PDP-1d. By the time the 1970 paper was written, we were a year beyond assembling on the Honeywell machine. Leaving that language in the paper, taken from a quarterly report to ARPA, was an oversight.
19. A.A. McKenzie et al., "The Network Control Center for the ARPA Network," *Proc. 1st Int'l Conf. Computer Comm.*, S. Winkler, ed., 1972, pp. 185–191; http://walden-family.com/impcode/iccc-1972-nmc.pdf.
20. Network Control Center program listing, version 52, May 1971, written by John McQuillan and in 2013 OCR'ed by James Markevitch from a scan by John from his archives; http://walden-family.com/impcode/ncc52 and http://walden-family.com/impcode/ncc52conc.txt.
21. A. McKenzie and D. Walden, "The ARPANET, the Defense Data Network, and the Internet," *Encyclopedia of Telecommunications*, vol. 1, Marcel Dekker, 1994, pp. 341–376; http://walden-family.com/public/encyclopedia-article.pdf.
22. BBN, *COINS Support and Maintenance Guide*, BBN TM-CC-0344, undated.
23. Honeywell's X16 series of computers also included the 116, 316, 416, and 716.
24. N.J. Liaaen and D.C. Walden, "Remembering the LFK Network," *IEEE Annals of the History of Computing*, vol. 24, no. 3, 2002, pp. 79–81; http://walden-family.com/ieee/lfk-2002-annals.pdf.
25. S.M. Ornstein et al., "The Terminal IMP for the ARPA Computer Network," *AFIPS Conf. Proc.*, vol. 40, 1972, pp. 243–254.
26. J.M. McQuillan et al., "Improvements in the Design and Performance of the ARPA Network," *Proc. AFIPS Fall Joint Computer Conf.*, 1972, pp. 741–754; http://walden-family.com/impcode/1972-improvements-paper.pdf.
27. R.E. Kahn and W.R. Crowther, *A Study of the ARPA Network Design and Performance*, BBN report 2161, Aug. 1971; http://walden-family.com/impcode/1971-Kahn-Crowther-performance-study.pdf.
28. R.E. Kahn and W.R. Crowther, "Flow Control in a Resource Sharing Computer Network," *Proc. 2nd ACM IEEE Symp. Problems in the Optimization of Data Comm. Systems*, 1971, pp. 108–116; http://walden-family.com/impcode/1971-Kahn-Crowther-performance-study.pdf.
29. J.M. McQuillan to D. Walden, email, 5 Aug. 2013.
30. J.M. McQuillan, *Adaptive Routing Algorithms for Distributed Computer Networks*, BBN report 2831, 1 May 1974. (This was also McQuillan's PhD thesis.)
31. J.M. McQuillan, I. Richer, and E.C. Rosen, "The New Routing Algorithm for the ARPANET," *IEEE Trans. Comm.*, vol. 28, no. 5, May 1980, pp. 711–719,
32. J.M. McQuillan, "The Birth of Link-State Routing," *IEEE Annals of the History of Computing*, Jan.-Mar. 2009, pp. 68–71.

33. See http://en.wikipedia.org/wiki/Open_Shortest_Path_First.
34. F.E. Heart et al., "A New Minicomputer/Multiprocessor for the ARPA Network," *AFIPS Conf. Proc.*, vol. 42, 1973, pp. 529–537.
35. S.M. Ornstein et al., "PLURIBUS—A Reliable Multiprocessor," *AFIPS Conf. Proc.*, vol. 44, 1975, pp. 551–559.
36. D. Katsuki et al., "Pluribus—An Operational Fault-Tolerant Multiprocessor," *Proc. IEEE*, vol. 66, no. 10, 1978, pp. 1146–1159.
37. D. Walden and R. Nickerson, eds., *A Culture of Innovation: Insider Accounts of Computing and Culture at BBN*, Waterside Press, 2011, pp. 534–538; http://walden-family.com/bbn/.
38. M.F. Kraley et al., "Design of a User-Microprogrammable Building Block," *Proc. 13th Ann. Microprogramming Workshop, Colorado Springs*, 1980, pp. 106–114.
39. Walden and Nickerson, eds., *A Culture of Innovation*, pp. 527–528.
40. BBN, *C/30 PSN X.25 Interface Specification*, BBN report 5500, release 3, 1 Nov. 1983.
41. BBN, *Quarterly Technical Report—TAC Functional Specification*, BBN report 4401, 1 June 1980.
42. BBN, *TAC Users' Guide*, BBN report 4780, 1 Oct. 1982.
43. BBN, *PAD Performance Evaluation*, BBN-TM-CC-0267, 21 Oct. 1986.
44. J. Postel, *NCP/TCP Transition Plan*, IETF RFC 801, Nov. 1981, http://tools.ietf.org/html/rfc801
45. BBN, *C/30 Native Mode Firmware System Programmer's Reference Manual, C/30E NMFS*, BBN report 5000, rev. 2, Microcode Version m7u13, 1 Aug. 1984.
46. BBN, *Release Reference Manual: C/300 Update Software*, BBN report 6289, Mar. 1986.
47. Walden and Nickerson, eds., *A Culture of Innovation.*
48. J. Markevitch, to D. Walden, emails, 21 Sept. and 22 Oct. 2013.
49. PDP-1 Midas was used to assemble IMP program from 1969 on, as described in the "Preparation, Implementation, and Installation" section.
50. J. Markevitch, "midas516.pl assembler for the Honeywell DDP-516/316 computers," Perl listing of Midas for the ARPANET 516 IMP and NCC programs, 2013; http://walden-family.com/impcode/midas516.txt.
51. IMP 3050 program file, as created by J. Markevitch from his OCR of the IMP 3050 listing scan, http://walden-family.com/impcode/imp3050.txt
52. Makefile from J. Markevitch that runs the IMP 3050 program file through his Midas assembler (see the prior two entries in this list); http://walden-family.com/impcode/, http://walden-family.com/bbn/Makefile.txt
53. C. Neuhauser to D. Walden, email, 30 Aug. 2013.
54. See http://walden-family.com/impcode/neuhauser-summary.pdf.
55. Developed by Robert Supnik, simh is a computer emulation system that runs on many different platforms with various operating systems and has been used as the basis for emulation of many different computers. It is widely used by people working to preserve software for computers that are no longer made. See http://simh.trailing-edge.com/ and https://github.com/simh/simh.
56. R.M. Supnik, "H316 Simulator Usage," 1 Dec. 2008; http://simh.trailing-edge.com/pdf/h316_doc.pdf.
57. A listing of demonstration software for the original 1973 ArpaNET IMP is available at http://simh.trailing-edge.com/software.html.
58. R. Armstrong to D. Walden, email, 31 Aug. 2013.
59. The Interface Message Processor, BBN TIR89, February 1973, http://walden-family.com/impcode/IMPSYS-Document-with-flowcharts.pdf.
60. BBN, *The Interface Message Process*, BBN report TIR89, update of Nov. 1973; http://walden-family.com/impcode/Technical_Information_Report_89.pdf.
61. IMP program listing, version 2514, Dec. 1971, OCR'ed in 2013 by Markevitch from a scan by McQuillan from his archives; http://walden-family.com/impcode/mp2514.txt and http://walden-family.com/impcode/imp2514conc.txt.
62. IMP program listing, version 3147, July 1974, OCR'ed in 2013 by Markevitch from a scan by McQuillan from his archives, http://walden-family.com/impcode/imp3147.txt, http://walden-family.com/impcode/imp3147malloc.txt, and http://walden-family.com/impcode/imp3147patch.txt.

David Walden retired from BBN in 1995. At BBN he had been one of the three computer programmers who originally developed the Arpanet IMP computer software. In retirement, he has been an IEEE Computer Society volunteer in the area of computing history; see http://walden-family.com/ieee/, which also includes contact information for him.