The ARPANET IMP Program: Retrospective and Resurrection

The IMP Software Guys

draft of December 2, 2013

The ARPANET technology has been extensively documented by BBN people [Heart94] and others. Sections 1, 2 and 3 of this paper sketch the history (not the inherent technology which has previously been described in many publications) of the ARPANET Interface Message Processor (IMP) program as originally written in 1969 for the modified Honeywell 516 computer. Other systems, derived more or less from the original system and running on a variety of hardware platforms, are also enumerated.

In 2013 a faded 1973 line printer listing of the IMP program from 1973 was run through a special 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 IMP hardware platform was created. Sections 4 and 5 of the paper describe the methods used to recover a digital copy and assemble and run again the 1973 IMP code.

* * *

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 paper, and thus we think it is an appropriate homage for the paper's author to be "the IMP Software Guys." We include within this designation the non-BBN people from Silicon Valley who, in 2012–2013 participated in the resurrection of the original 516 IMP program.

[The following table of contents is only here to assist reviewers in assessing the structure of the paper. It will be removed for the published version of the paper.]

Contents

Part I: ARPANET IMP Program Retrospective

1 Preparation, implementation, and installation, 1968-1970

In 1968 BBN was preparing to bid on the contract from ARPA to develop the ARPANET Interface Message Processors (IMPs) [ARPA68]. 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 [BBN68] to ARPA, Will Crowther was added to the team as another (more senior) software person; and 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 to IMP development contract with a start date of January 1, 1969.

Over 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 IMP-to-host code. Nonetheless, all three knew the entire software system inside out [BBN Report 1763, BBN Report 1822, BBN Report 1877]. ("Host" is the name for computers connected to an IMP and using the network of IMPs to communicate with others host computers. By this definition, a personal computer connected today to a router or a company web server connected to a router are host computers, using the internetwork of routers to communicate with other computers.)

The IMP hardware was a modified Honeywell 516 [Honeywell1, Honeywell2, Honeywell3]. However, the IMP software was developed on BBN's PDP-1d using the TECO editor [Murphy09] for composing and editing the program and the PDP-1d's Midas assembler [Midas1, Midas2] modified to understand the Honeywell 516 instruction codes, word size, and page boundaries. The assembled program in octal was then output on paper tape for loading into the IMP via its paper tape reader; after BBN's PDP-1d was connected to the ARPANET in 1971, new IMP software could be loaded into IMPs via the network itself. (An aside: The original paper about the IMP from the BBN developers [Heart70] 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 very 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.)

Starting around Labor Day in 1969, one IMP a month was delivered in succession to UCLA, SRI, UCSB, and the University of Utah. Early in 1970 a fifth IMP was installed at BBN, and a Network Control Center capability was developed by late 1971 [McKenzie72, NCCv52].

By way of context, the Honeywell 516 computer had 32 thousand bytes of random access memory and a 1 microsecond cycle time (and was the size of a refrigerator). A random laptop computer (say in the Toshiba Satellite line) in 2013 has 4 billionF bytes of random access memory (a factor of 125 thousand bigger than the IMP's memory) and a cycle time 2.8 thousand times as fast (although it is hard to make a valid cycle and instruction time comparison given how different computer architectures are now versus then). It is hard for some people to imagine today how a entire packet switching system could be implemented in a computer as small as the 516 IMP.

* * *

The ARPANET, which became operational as a research network in early 1970, directly transi-

tioned to or somewhat influenced a number of other networks. We will mention these other networks in Section 2. In Section 3 we will provide a sketch of how the Honeywell-516-based IMP code evolved for use on several different hardware platforms (the same code also ran on the non-hardened Honeywell 316, also modified with BBN designed interfaces to support the packet-switching function).

2 Evolution of the ARPANET and its successors and derivatives

Between 1969 and 1975 the ARPANET expanded at the rate of about 10 IMP per years (to approximately 60 IMPs total). Over the period from July 1 to December 31, 1975, the ARPANET operation was phased over from ARPA to the Defence Communications Agenca (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 made the decision that the ARPANET would be the foundation of the Defense Data Network (DDN). Creation of DDN involved splitting the existing ARPANET into two parts — a part which was still known as ARPANET and served non-military users, and a part which was known as MILNET and served military users. (The two parts could communicate in a controlled way by devices called mail bridges.) Also over this period, the ARPANET (in its pre-DDN and DDN form) 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 [McKenzie94].

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 U.S. intelligence community [COINS]. In 1972 BBN organized a commercial packet-based telecommunications carrier known as Telenet which originally used a version of the 516 IMP code running on Honeywell 716 computers but later built their own computer and redid the software. BBN made consulting agreements with Logica in the U.K. and SESA in France for them to bid the 516/316 IMP technology, 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) which original handled the hardware maintenance contracts for BBN-delivered networks (as well as trying to be a mini-computer vendor more generally). By 1982, the name of BBNCC was changed to BBN Communications Corporation and its activities were refocused on networking. From this part of the company, BBN delivered many commercial, military and other networks around the world.[1] BBNCC supported the 516/316 IMP technology, the Pluribus IMP technology, and developed new IMP technology based on BBN's C/30 and C/300 computers. An overview of these various systems is shown in Figure 1.

BBN's ARPANET technology also influenced the design of several non-BBN networks. The LFK Network was a deliberate copy of the ARPANET code running on Norsk Data computers and rewriting the IMP code from scratch [Liaaen02]. The founders of Packet Communications Corporation (who left BBN a little before Telenet was founded) had a copy of the IMP system

[1] [The following footnote text can be dropped from this paper and listed on a website for the paper.]
On October 4 and 7, 2013, an exchange of emails on the ex-BBN mailing list listed some of the companies to which BBNCC and other parts of BBN delivered networks — among them ARPANET/MILNET Abbey Bank, Amadeus, BBN Networks, BBN net, Bank of Hawaii, Barclays, Berkeley Library, Burlington Northern, Chembank, Citibank, Customs/Treasury, DISNET 1,2, 3, ENI, 5th Signal Net, ISTEL, ITT, IRRC, Irving Trust, JAL, KDD, MCI, MINET, Many military exercise networks (Reforger, Forger, etc.), Marriot, Mastercard, Michigan Bell, NKK, NatWest, Nexis, On-Line Systems, PGI, Packet Radio Net, Packet Voice Satcom, Schulumberger, Sixtel, USAF Academy, VISA, Wang, Weyerhauser, WINNET, Wideband net, and several classified networks. Contributors to this exchange were Jack Haverty, Bernie Cosell, Ken Turkewitz, Ben Barker, Keesan, Cliff Romash, and, with particular good memories or personal archives, Alan Hill and Peg Primak.

	516/316	Pluribus	C/30 (MBB with IMP software)	C/30 PSN (IMP+X.25)	C/300 PSN using NFS	T/300 (68020) = T/100 hardware + C/300 software
Initial deployment	1969	Initial design ~ Q1 1972 Fielded ??	1979	1982	1984	ca.1988-90
Packet switch	IMP	Pluribus IMP (HSMIMP) hardware operational Q3 1974	IMP	PSN	IMP or PSN (not both)	PSN (only deployed commercially)
Terminal concentrator	TIP begun Q3 1970; field operation in Q3 1971	Pluribus TIP (PTIP) Hardware operational by Q3 1974	TAC	PAD (based on C/50 - a C/70 without disk)	ditto?	ditto?
Satellite connection	x16 able to handle satellite circuits until replaced by Satellite IMPs using TDMA, slotted Aloha, pure Aloha	Pluribus Satellite IMP (PSAT) which could (perhaps?) interact with x16s; later CPODA only on PSATs				
Network operations center	Data collection on 316 (with laster support on PDP-1d and TENEX		NOC also moved to MBB	Eventually NOC moved to C/70 Unix NU		
Software development	PDP-1 (also for debugging via IMP fake host DDT and for sending new releases	Assembly moved in time to PDP-10				

Figure 1: IMP hardware

listing (BBN was required to give copies to companies which asked for a slightly 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 (e.g., the Internet): dynamic routing, operation without central control, packet flow control and reliable transmission, its operating philosophy, its engineering approach to standards, etc.).

3 Evolution of the IMP code

A main point of this paper 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 the course of many years; see Figure 2.

Making the network algorithms really work, on the 516/316

After the initial IMP installations in 1969, the ARPANET continued to expand and, being a more or less operational network, fixes were required and improvements could be made to the 516/316 IMP software [McQuillan72]. Later the algorithms were reimplemented so Pluribus-platform-based IMPs could function as an IMP 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 were developed on a platform other than the 516/316 IMP platform.

Here we will sketch the initial round of changes to the IMP software by quoting from a 1972 paper [McQuillan72].

> 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 [BBN Report 2161, Kahn71]. 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 in the following subsections.
>
> 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.

Year	Function
1969	Original IMP code
Begun Q3 1970; first field operation Q3 1971	TIP code
Design begun Q4 1971; operations ~Q1 1972	VDH code
1971-1974	Fix IMP end-to-end bugs, improve modularity (different speed, TIP, VDH, robustness, throughput)
1977-1979	Completely replace routing code (link-state/SPF)
ca. 1980-1981	IMP code runs on MBB-base C/30 emulating 316
ca. 1982	X.25 added
??	TAC
??	PAD
ca. 1981-1983	NCP-to-TCP transition
ca. 1984	NMFS developed converting MBB into a real-time extended memory machine independent of the software application; IMP code adapted to run on NMFS version of C/30
ca. 1983-1985	IMP code adapted to run on C/300
1985-1987	End-to-end code for IMP redone to embed X.25
1987-1989	Store-and-forward code for IMP redone to improve conjestion control

Figure 2: Software evolution

> 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 a bit of wiring away from the IMP], or TIP hardware [an IMP option for directly connecting to it a software host handling 63 terminals], 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 [McQuillan13].

> ... [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 above changes were made, the major effort of the next few years was redoing the original ARPANET routing as the limitations of the original algorithm were discovered as the network grew larger. First there were little modifications, and then McQuillan looked at the issues in detail [McQuillan74]. Eventually, McQuillan and others developed a new routing algorithm [McQuillan80, McQuillan09] which the then 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 OSPF routing algorithms [McQuillan09]. Although not unique to the routing transition, the routing transition was an instance where 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 above evolution of the 516/316 IMP software was happening, the Pluribus multiprocessor system was being developed and the IMP algorithms were recoded for it [Heart73, Ornstein75]. 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 [Katsuki78, Walden11 pp. 534-538]. This was so successful there was no simple off switch for the machine; a program had to be run to shut parts of the machine down faster than the machine could "fix itself" and keep running.

The Pluribus IMPs and 516/316 IMPs ran together and compatibly in the ARPANET and later in DDN. In particular, the complex transition to the new ARPANET routing algorithm mentioned at the end of the prior section has to be done in the 516/316 IMP and Pluribus IMP in parallel.

MBB-based systems

In 1978 BBN developed its Micro-programmable Building Block (MBB) computer which could be coded by users to perform different functions [Kraley80, Walden11 pages 527–528]. 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 micro code. (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**[check]** and to run network-control-center software.) Approaching 1,500 MBB-based systems were installed in networks delivered by BBN. Most of this work was done in BBN Communications Corporation (see page 3).

The efforts in the early years of BBNCC primarily consisted of adding capabilities to the edges of a C/30 network. One such effort was adding (ca. 1982) to the IMP software an interface for X.25 hosts; the X.25 interface initially ran on top of the standard ARPANET 1822 host interface [BBN Report 5500]. This system simultaneously supported the ARPANET 1822 and X.25 interfaces. At the time**[check]** the X.25 interface was added, BBN Communications Corporation began calling the system a PSN**[check]**. BBNCC also developed new terminal concentrators for this line of packet switches, called TACs (ca. 1980–1982) [BBN Report 4401, BBN Report 4780] and PADs (1981–1983) which were terminal concentrators in a computer separate from the packet switch [BBN TM-CC-0267].

During the 1981–1983 period, BBNCC also had to deal with the conversion of the networks it was operating from the ARPANET NCP host-host protocol to use by the hosts of TCP/IP. BBN had to adapt its own systems functioning as hosts to TCP/IP and was involved in the staging of the transition for other hosts [RFC810,**[check]**is there an RFC by a BBN author about the transition?, .].

In 1982–1983, the IMP code underwent a major change as new microcode (known as the Native Mode Firmware System — NMFS) was written for the MBB which 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 done in microcode, and some additional instructions for queuing functions) [BBN Report 5000].

In 1983–1985, a new hardware machine was built, the C/300. This was a faster MBB, but with the customizations for the Honeywell emulation built in, so that it could not run in C/70 or other hardware modes [BBN Report 6289]. This was the workhorse in the DDN upgrade.

Note that many of the above MBB based developments concerned running the IMP code 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 it riding on top of the ARPANET’s 1822 interface.**[citation?]** This was the PSN 7 version of the system for which a listing is known to exist.**[citation?]**

In 1987–1989, another major change to the basic packet-switching algorithms was done: the packet-switching store-and-forward code was redone to include congestion control.**[citation?]**

This was the PSN 8 version of the system for which a listing is known to exist.**[citation?]**

* * *

In these first three sections we have sketched the way the original ARPANET IMP code was reused, reimplemented, extended, and changed over more than 30 years. We have also hinted at the impact the original ARPANET IMP functions on numerous more or less closely related networks — certainly to some extent in the Internet today.

Part II: ARPANET IMP Program Resurrection

4 Recovery of 1973 516/316 IMP system listing, 2002–2013

In 2002 Paul Wexelblatt, who had been a member for BBN's ARPANET IMP team at one point, was cleaning out the basement of his house. He queried Bernie Cosell and Dave Walden about 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, would keep these for a while as he worked on the BBN computing history book [Walden11], and then would offer anything Cosell wanted to him for permanent possession. In particular, Cosell asked that the IMP program listing be passed to him, and eventually it was.

In 2009 Tony Michel 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 more was needed, and he passed the listing to Walden to work on some more scanning.

In 2012 Jack Haverty (who had worked at BBN, close to the IMP system, early in his career) 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 OCRing it to recover a source code file which could perhaps be assembled and run on simulated IMP machines. Walden still had the listing in hand and offered to take the listing to the Boston office of the law firm for which Haverty was consulting to get a good quality scan done, and the scan was done.

However, getting decent OCR from the scan was not successful. Several different OCR programs were tried and none produced good results. Still Walden posted the scan of the listing on his website with other historical IMP system content [IMP73].

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

In parallel James Markevitch became aware of the scan of the IMP system listing and ran it through his home-built OCR program optimized for faint line printer listings. James explained as follows [Markevitch13]:

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 1950's) 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 feed 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 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 [Midas3].

I have done this same sort of thing numerous times: OCR a listing, convert to a file that can be input to an assembler [Midas4], write an assembler, assemble, compare the output with the OCR'd listing, and iterate [Midas5] 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.

* * *

All of the scans and listings mentioned above and below are available for the reader to peruse at `walden-family.com/bbn/#ref-impsoftware`

* * *

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. Below is Neuhaurser's story [Neuhauser13a]:

> Initially, I went through the entire listing and extracted all the IO opcodes. From this, the Honeywell manuals [Honeywell1, Honeywell2] 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 [BBN68] 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 [Neuhauser13b]. 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 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 [IMP73], 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 PDP-8**[check]** 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` [Simh] 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.

5 IMP software cycles again, 2013

As mentioned above, Bob Armstrong built the simulator for the 516/316 IMP. He started with the existing Honeywell 316 simulator [Simh316] based on the `simh` simulator [Simh], and then he modified the H316 simulator to simulate the instructions and interfaces BBN added to the 316 computer to run the IMP system [Honeywell3] as part of the IMP system development [BBN Report 1763, BBN Report 1822, Heart70]. The software kit for Bob's IMP simulator (released to the public courtesy of the aforementioned, but anonymous, law firm) is available on the web [SimhIMP].

Bob Armstrong's version of the story is below [Armstrong13]:

> 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 [Neuhauser13b]. Later on the BBN IMP guys found and passed along more period documentation from BBN [BBN TIR 89a, BBN TIR 89b] 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.

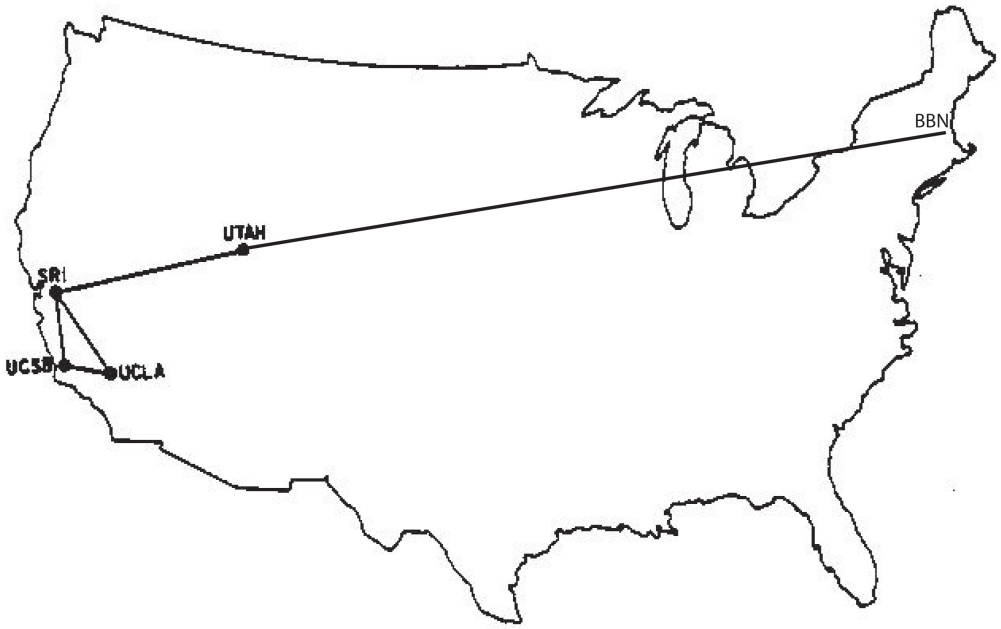

Figure 3: Early 1970 ARPANET map [a crisper version of this map can be created]

> 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 the store-and-forward worked, one could use the simulated Teletype of one IMP to inspect and change memory of another IMP, 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 July 5 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 4- and 5-node ARPANET configurations in 1969 and early 1970 (Figure 3): 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'd them, making needed additions to his Midas IMP-code assembler as necessary, and these listings also have been posted on the web [NCCv52, IMP2514, IMP3147].

* * *

The first three sections of this note are a relatively straightforward historical account, albeit an extended anecdote collected from many people rather than a formal piece of computing history research. The last two sections describe an effort that is part of a relatively new area of computing history work (that some are calling "living history" and others call "retro history") in which artifacts from computer history are brought back to life. This latter part of the story may not be done: We plan to continue the effort to collect, organize, and make such IMP artifacts available to the computing history world.

Participants

[This following list of people can be moved to a website and only list the URL of the participants website here.]

* * *

Among the people who were involved with the packet switch code or this recounting of its history were Len Abram, Ellie Baker, Ben Barker, Rosemary Carter, Bernie Cosell, Will Crowther, Peter Cudhea, Jim Dempsey, Eric Elsam, Walter Gillett, Jack Haverty, Jim Herman, Alan Hill, Bob Hinden, Doug Hirsch, Richard Koolish, Mike Kraley, Ken Laube, Joel Levin, Ben Littauer, Andy Malis, Alex McKenzie, John McQuillan, Tony Michel, Drew Powles, Peg Primark, Ira Richer, Eric Roberts, John Robinson, Cliff Romash, Eric Rosen, Paul Santos, Ken Turkewitz, Dave Walden, Jil Westcott, and Paul Wexelblat. No doubt I have missed some people, and these can be added to the on-line list at `walden-family.com/bbn/imp-code`

In addition to some of the people listed in the prior paragraph, the people involved in the 2013 recovery of the IMP code were: Bob Armstrong, Tom Kilbourn, James Markevitch, and Charlie Neuhauser, who reside and work in Silicon Valley, and BBN librarian Jennie Connolly.

Acknowledgments

As noted in the abstract, many people contributed to this paper and to the work described herein. Most of them are probably listed in the prior section [or on the website that goes with this paper if the previous section is dropped from the published version of the paper]. No doubt some people have also been left off the list; we thank them too.

Andy Russell and Jim Cortada gave suggestions for how this paper might be restructured (e.g., break it into two papers, or mostly replace sections 1 and 2 with a short summary of their current content). We also thank the anonymous referees. [If there is positive feedback from the *Annals* review process about potential acceptance of this paper, we will act appropriately on the various bits of advice. If the *Annals* is uninterested in this paper, we will probably keep it more or less as it is and publish it elsewhere.]

Jason Armistead and David Gesswein pointed out an error in the paper.

Bibliography

A Culture of Innovation: Insider Accounts of Computing and Culture at BBN,
David Walden and Raymond Nickerson (eds.), Waterside Press, East Sandwich, MA, 2011.

http://walden-family.com/bbn/#cultureinnovation

ARPANET and Internet: A Bibliography of BBN Papers, September 1994.
http://walden-family.com/bbn/ArpanetInternetBiblio.pdf
This bibliography was created for the ARPANET 25th anniversary celebration organized by BBN and held in Boston.

Matthias Bärwolff collection of BBN networking reports and report excerpts.
http://xn--brwolff-5wa.de/bbn-arpanet-reports-collection/
Ultimately we hope to have available a more comprehensive collection of BBN reports.

Networking at BBN website.
http://walden-family.com/bbn/#networking

References

[We are still looking for some of the URLs needed below. If you have any, please pass them along.]

Armstrong13. Robert Armstrong, email of August 31, 2013.

ARPA68. ARPANET RFQ, walden-family.com/bbn/arpanet-rfq.pdf

BBN68. BBN's IMP proposal, walden-family.com/bbn/arpanet-prop-ocr.pdf

BBN Report 1763. Initial Design for Interface Message Processors for the ARPA Computer Network, January 1, 1969, URLxxx

BBN Report 1822. Interface Message Processor: Specifications for the Interconnection of a Host and and IMP, original edition by Robert E. Kahn, May 1, 1969; this report was revised a number of times over the years, for example, URLxxx

BBN Report 2161. R. E. Kahn and W. R. Crowther, A study of the ARPA network design and performance, August 1971, URLxxxx

BBN Report 4401. Quarterly Technical Report — TAC Functional Specification, June 1, 1980.

BBN Report 4780. TAC Users' Guide, October 1, 1982.

BBN Report 5000. "C/30 Native Mode Firmware System; Programmer's Reference Manual; C/30E NMFS; Rev. 2, Microcode Version m7u13," August 1, 1984.

BBN Report 5500. C/30 PSN X.25 Interface Specification, Release 3, November 1, 1983.

BBN TIR 89a. The Interface Message Processor, BBN TIR89, February 1973, walden-family.com/bbn/IMPSYS-Document-with-flowcharts.pdf

BBN TIR 89b. The Interface Message Process, BBN TIR89, update of November 1973, walden-family.com/bbn/Technical_Information_Report_89.pdf

BBNCC TM-CC-0267. PAD Performance Evaluation, October 21, 1986.

Heart70. F. E. Heart, R. E. Kahn, S. M. Ornstein, W. R. Crowther, and D. C. Walden, The Interface Message Processor for the ARPA Computer Network. *AFIPS Conference Proceedings*, vol. 36, pp. 551-567. AFIPS Press, Montvale, NJ, May 1970.

Heart73. F. E. Heart, S. M. Ornstein, W. R. Crowther, and W. B. Barker, A New Minicomputer/ Multiprocessor for the ARPA Network, *AFIPS Conference Proceedings*, vol. 42, AFIPS Press, Montvale, NJ, June 1973, pp. 529–537.

Heart94. ARPANET and Internet: A Bibliography of BBN Papers, September 1994.

Honeywell1. Honeywell 316 Computer Hardware, walden-family.com/bbn/70130072176C_316_CPU_Descr_Apr73.pdf

Honeywell2. Honeywell 316 Programmers' Reference Manual, walden-family.com/bbn/70130072156_316_516_PgmrRef_Nov70.pdf

Honeywell3. Honeywill modifications manual for the IMP (mistitled), walden-family.com/bbn/imp-hardware.pdf

IMP73. walden-family.com/bbn/#ref-impsoftware

IMP2514. IMP program listing, version 2514, December 1971, OCRed in 2013 by James Markevitch from a scan by John McQuillan from his archives, walden-family.com/bbn/mp2514.txt and walden-family.com/bbn/imp2514conc.txt

IMP3147. IMP program listing, version 3147, July 1974, OCRed in 2013 by James Markevitch from a scan by John McQuillan from his archives, walden-family.com/bbn/imp3147.txt and walden-family.com/bbn/imp3147malloc.txt and walden-family.com/bbn/imp3147patch.txt

Kahn71. R. E. Kahn and W. R. Crowther, Flow control in a resource sharing computer network, *Proceedings of the Second ACM IEEE Symposium on Problems in the Optimization of Data Communications Systems*, Palo Alto, CA, October 1971 pp. 108–116.

Katsuki78. D. Katsuki, E. S. Elsam, W. F. Mann, E. S. Roberts, J. G. Robinson, F. S. Skowronski, and E. W. Wolf, Pluribus — An Operational Fault-Tolerant Multiprocessor, *Proceedings of the IEEE*, vol. 66, no. 10, 1978, pp. 1146–1159.

Kraley80. M. F. Kraley, R. D. Rettberg, P. Herman, R. D. Bressler, and A. Lake, Design of a User-Microprogrammable Building Block, *Proceedings of the 13th Annual Microprogramming Workshop, Colorado Springs, CO*, IEEE, New York, NY, December 1980, pp. 106–114.

Liaaen02. Nils J. Liaaen and David C. Walden, Remembering the LFK Network, *IEEE Annals of the History of Computing*, Vol. 24, No. 3, July-September 2002, pp. 79–81, http://walden-family.com/ieee/lfk-2002-annals.pdf

Markevitch13. James Markevitch, emails of September 21 and October 22, 2013.

McKenzie72. A. A/ McKenzie, B. P. Cosell, J. M. McQuillan, and M. J. Thrope, The Network Control Center for the ARPA Network, *Proceedings of the First International Conference on Computer Communication*, Winkler (Ed.), Washington, DC, October 1972, pp. 185–191, walden-family.com/bbn/icccc-1972-nmc/index.html

McKenzie94. Alexander McKenzie and David Walden, The ARPANET, the Defense Data Network, and the Internet, *Encyclopedia of Telecommunications*, Marcel Dekker, Inc., Volume 1, pp. 341–376, 1994, http://walden-family.com/public/encyclopedia-article.pdf

McQuillan72. J. M. McQuillan, W. R. Crowther, B. P. Cosell, D. C. Walden, and F. E. Heart, Improvements in the design and performance of the ARPA network, *Proceedings of the AFIPS Fall Joint Computer Conference*, 1972, pp. 741-754.

McQuillan74. John M. McQuilla, Adaptive Routing Algorithms for Distributed Computer Networks (his PhD thesis), BBN report 2831, May 1, 1974.

McQuillan800. J. M. McQuillan, I. Richer, and E. C. Rosen, The New Routing Algorithm for the ARPANET, *IEEE Transactions on Communications*, Vol. COM-28, No. 5, May 1980.

McQuillan09. John M. McQuillan, The Birth of Link-state Routing, *IEEE Annals of the History of Computing*, January-March 2009, pp. 68–71.

McQuillan13. John McQuillan, email of August 5, 2013.

Midas1. Robert A. Saunders et. al, MIT PDP-1 Midas memo, date unknown, http://archive.org/details/bitsavers_mitrlepdp1_1535627

Midas2. BBN PDP-1 Midas manual, *Memorandum Number 6-E: Programming Software Status Report*, Hospital Computer Project, Bolt Beranek and Newman Inc., Cambridge, MA, May 1, 1966, walden-family.com/bbn/bbn-1966-pdp1d-midas-manual.pdf

Midas3. James Markevitch, Perl listing of Midas for the ARPANET 516 IMP and NCC programs, 2013, walden-family.com/bbn/midas516.txt

Midas4. IMP 3050 program file, as created by James Markevitch from his OCR of the IMP 3050 listing scan, walden-family.com/bbn/imp3050.txt

Midas5. Makefile from James Markevitch which runs the IMP 3050 program file through his Midas assembler (see the prior two entries in this list), walden-family.com/bbn/http://walden-family.com/bbn/Makefile.txt

Murphy09. murphy on Teco, *IEEE Annals of the History of Computer*, vol. , no. , 2009, pp. .

NCCv52. Network Control Center program listing, version 52, May 1971, written by John McQuillan and in 2013 OCRed by James Markevitch from a scan by John from his archives, walden-family.com/bbn/URL

Neuhauser13a. Charles Neuhauser, email of Auguest 30, 2013.

Neuhauser13b. Charles Neuhauser's list of the IMP system operation codes, derived by studying the scan of the assembly listing, URL???

Ornstein75. S. M. Ornstein, W. R. Crowther, M. F. Kraley, R. D. Bressler, A. Michel, and F. E. Heart, PLURIBUS — A Reliable Multiprocessorm AFIPS Conference Proceedings, vol. 44, AFIPS Press, Montvale, NJ, May 1975m pp. 551–559.

RFC810. Jon Postel, NCP/TCP Transition Plan, November 1981, `http://tools.ietf.org/html/rfc801`

Simh. Robert M. Supnik, et al., `http://simh.trailing-edge.com/`

Simh316. Robert M. Supnik,`http://simh.trailing-edge.com/pdf/h316_doc.pdf`

SimhIMP. Robert Armstrong, Demonstration software for the original 1973 ArpaNET IMP, at `http://simh.trailing-edge.com/software.html`

Walden11. *A Culture of Innovation: Insider Accounts of Computing and Culture at BBN*, David Walden and Raymond Nickerson (eds.), Waterside Press, East Sandwich, MA, 2011, `walden-family.com/bbn/`