Interview of Stuart “Stu” Wecker

Interviewed by:
James Pelkey

Recorded: October 25, 1988
Sudbury, MA

CHM Reference number: X5671.2010

James Pelkey: Maybe you might just start with your background, how you got to DEC and to the point where things were starting to shape your thinking when you were posed with the opportunity to get involved with and create DECnet.

Stuart “Stu” Wecker: Ok. I think that's sort of an interesting story, a little bit. My background is computer software. I hold a Bachelor of Science degree in Mathematics from RPI 1966. I'll actually give you a short, one page bio.

Pelkey: I'm RPI '68.

Wecker: RPI '68? Excellent, a fellow alumnus. I got a Master of Science in Computer Science from State University of New York, Stony Brook, where I was working -- prior to DEC; I was manager of system at the computing center, and in those days, working on your typical IBM mainframe. In those days, we had an IBM 360 Model 67, which, if you recall, was the timesharing computer, the one that TSS ran on. Those were in the early days when IBM had a research project going on in Cambridge, Mass., called CP-67. It was the forerunner of VM-370. The concept was to build a virtual machine, of which the Model 67 was actually able to emulate, in user space, a Model 65, in essence, or a vanilla Model 360. I worked for a number of years, on a number of projects that related to CP-67. I was in the measurement group, I belonged to SHARE, the users' group, and I was influential in the measurement subcommittee, and I actually created, in the end, a number of concepts, including one called the preferred virtual machine, which ran one virtual machine at fairly high performance levels. I eventually incorporated that into VM-370.

Pelkey: And this is what years,'68 to '70-ish?

Wecker: This is in the years of '68 to, like, the beginning of '71. What happened at that time, I guess I was getting itchy and sort of ready to move on. Stony Brook, at that time, had just gotten a DEC System 10. I had never even, at that point believe it or not, ever heard of the name DEC at all, and they were just moving it in when I was anxious to look at a job change, and I actually noticed an ad in, I think it was, the Sunday Times looking for a researcher in a DEC R&D group, somebody to do work on multiple processor minicomputer systems, somebody to look at ways to interconnect multiple minicomputers in some kind of a structure where they could achieve the power of large systems. I had never done work in that area, but I had done a lot of work in operating systems, and I thought it sounded intriguing, so I came up to Maynard and had a couple of interviews, and joined the research group.

Pelkey: In what year?

Wecker: That was 1972, June of '72. At that time, where DEC was at, I guess the PDP-1120, the first model of the 11, had already been out maybe about a year or so and was starting to sell, and DEC was in the design process for a number of new models of the 11, I think including the PDP-1105 and the 1140 as well. The project I set off to work at was how might you interconnect some set of systems in order to build a multi-processor. There had been other multi-processor systems built prior to that date by the large computer manufacturers, I think Burroughs probably being the leader of that back in the late '60s. Most of them were based on your traditional structure of some kind of a system bus and a shared memory and a number of processors all sharing that common memory, and some way to turn the operating system into some multi-system structure. I started looking at the problem, I started thinking about the problem, I started thinking about the way that processors start interacting in performing larger tasks, and came to the conclusion that the right way to integrate the processors together was in more of a loose coupling approach, more of a coupling approach where the processors could each run off asynchronously with respect to each other, and then call on each other when they needed to. I came upon the concept of the fact that the easy way to do this was to somehow link these processors together with some kind of a network fabric, but that, from a software point of view, that the ideal structure was some kind of a message passing, or message base, kind of a system. Today, we've seen a number of these come out, but this was 1971, '72. This was in the very early days when operating systems hadn't made a lot of progress in the message passing area. I think it's actually even possibly before the days before Dykstra had actually formalized semaphores as an actual construct.

Pelkey: Excuse me; you said you joined DEC in '72, right?

Wecker: '72, so here I was in late '72 thinking about this problem, and --

Pelkey: Had you been exposed to other message passing systems?

Wecker: No, I started reading some papers and had read a couple of small articles about people that had done message passing system, and the concepts, and thought about it a little bit and said: "That would seem to be the right structure. It would seem to be that if we could turn everything that happened in the computer into a set of messages between processes, then 1) we wouldn't care where the processes were located, we could make them location independent as long as we had a name space and as long as we had an addressing scheme where we could locate objects anywhere in this name space, we could pass messages back and forth between the processes. In fact, I even eliminated, from the software point of view, above the operating system, I totally eliminated the concept of interrupts, in that what would happen is the way you would actually write to an I/O device, you would actually send a message to an I/O device driver, and when the interrupt came back for completion, the device driver would process that from a hardware point of view, but what that would turn back into would be a message at the software level, which would get sent back to whichever process sent the requesting message. So the system was totally message based. I actually implemented this system.

Pelkey: Who were you bouncing these ideas off of at the time?

Wecker: The research group was a small group in DEC. We were approximately five of us. Jim Bell was the manager of the group. He has since moved on to Hewlett Packard. Bill [William D.] Strecker joined the group just after I did. He's well known today as the architect as the VAX, and I bounced a lot of these ideas off Gordon Bell as well, who had just come back, approximately a year or two before, from Carnegie Mellon, and he was now back in charge of DEC engineering, trying to work on the PDP-11 and new models of the 11. Research was never DEC's strong thing. In fact, research at DEC really, in general, meant more advanced engineering or advanced applied engineering. It never meant real abstract research. So we were a small group of five people, and so I went off and actually built a system, and did my own coding, PDP-11 assembler language. There wasn't any C at that time, the days before C. 'BC,' right. Before C. So, I coded everything in assembler language and actually built a little message-passing kernel. I built a little kernel of an operating system where you could spawn processes and processes had some number of message channels that they could send messages to other processes. They could then wait to get replies back, or they could continue to run immediately. They could send them off to I/O driver processes, but they didn't look any different from any other process, and I actually implemented a connection mechanism in a way to carry on a conversation between two processes. What was sort of interesting is that I hadn't read a thing about networking at this point. I didn't know what a network was yet, and yet I had actually designed my first transport protocol, even though I never knew the word 'transport protocol,' and in fact, in order to communicate with thing like disk drives, I had actually invented my first file access protocol, because I designed some message formats of how I spoke to this disk driver process and ask it for a block of data from the disk, and it sent me the block of data back, and so I had my first file access process, but I didn't know what to call it. In fact, I didn't even know the word 'protocol.' I had never used that word. To me, it was 'message passing.' I never thought of data networks much, not when I first did it, and I built this thing on a single processor base.

Pelkey: This would have been '73?

Wecker: This was about '73, or something. Then -- let's see -- two things happened; I’m not quite sure the order here, but I'm not sure that it's that relevant. The first thing that happened is that the PDP-11 had now come out as a machine, and one of the interesting things they had done in the PDP-11 -- sorry, let me back up a second. The PDP-1105 had just come out from DEC as their new PDP-11 processor. There was a scaled down and, I guess, more LSI version of the PDP-1120. It was a one-board processor, and one of the things that they did is they separated the bus arbitration logic from the processor functions, such that you could cut a jumper on the board and put a slave PDP-1105 processor on the Unibus, along with the master processor. The slave processor would still be able to access the memory, but would not become master of the bus, so that there was a single bus arbitrator that would remain in the main processor, and you could attach some number of slaves. Well here I saw was a hardware structure that could probably at least let me build my prototype. So I went and I attached three processors to a PDP-11 Unibus: one main processor, two slave processors, one large memory space, and a number of peripheral devices. The way it would work is that nay processor could actually start any peripheral device reading, but the interrupts would always come back to the main processor, because that's where the bus arbitration unit was. As bizarre as that sounds, it actually worked out fine in my software structure, because it didn't matter who fired off anything or where the interrupt came back, because eventually, it would all turn into messages that would get sent back to the requesting processes. I took my original system that ran on a single processor; the only change I had to make was in the system dispatcher, which would normally only dispatch a single process at a time. Now it had to keep track of the fact that there were multiples. Each processor could be running a process. Since it was all shared reusable code, all that had to happen is you just started up each -- after you started the main processor up, you would start the slaves up inside the dispatcher, where it would start looking for another runnable process. It literally was about 20 lines of code change in assembly language, wrote the thing up, and the first day it ran. It just ran.

Pelkey: That must have been an exciting day.

Wecker: It actually didn't surprise me all that much because the whole system had been designed to be a set of asynchronously executing processes anywhere. It was an operating system that had literally divorced itself from its hardware-based structure above the base kernel. Logically, the system ran great and I demo'ed it for Gordon and for a number of other people. Physically, the problem was that the PDP-1005 bus arbitration unit in the main processor was pitifully slow, and so every time one of those slave processors wanted to access memory, it had to go through an arbitration cycle of the main processor, and it, unfortunately -- that was the main problem, was that this hardware hadn't been designed to be a multi-processor, so unfortunately what happened was the thing ran -- two processors ran at about 1.5 time one, three processors ran at about 1.6 times run, and it just rapidly fell off so that after two or three processors, I hardly got any performance increase. I don't think I could ever get up to even two processors worth. The bus arbitration unit was 100% busy. It wasn't the software that was slowing this down at all. In fact, if I had had the right hardware base, I really believe that that thing would have gone very close to linear with the number of processors deployed. If I had gone up to two processors, I think it would have been about 1.9, three processors about 2.8, or what you would typically more expect, because there were no bottlenecks in the software. There was nothing in the software that was going to be a single bottleneck that was going to hold up the asynchronous operation. I demo'ed it and it was sort of cute, and put that system to bed.

Pelkey: Do you remember what time frame this was?

Wecker: This was in, I guess, the beginning of '74. The system was called SHARP. It stood for Stu's Homogenous Asynchronous Relocatable Process operating system, SHARP. The center of it, the little kernel, I called the CORE, for Communications Oriented Resident Executive, which is what it was. That was an interesting effort. I put that aside and, at the same time, since I had been doing work basically in that area -- what do I recall now -- two things started to happen. First, the DEC System 10, which was a fairly large-scale timesharing machine, had had a remote job entry station concept built into it some number of years before by the PDP-10 engineering people. I can't remember exactly how they engineered it and what they used, but it wasn't very high performing, and Gordon Bell decided that there was a need for a new remote job entry station to be built for the DEC system 10, and that we should use a PDP-11 for that function, to use a minicomputer to build our first little remote entry station that would have line printers and, I guess, even serve as a terminal concentrator. Gordon decided that a small committee should be put together with one representative from the PDP-11 engineering group, who was into communications, one representative from the DEC 10 engineering group, who had previously worked on their remote job entry station, and he picked me out of R&D because I seemed to be working somewhere related to the area of communications and protocols. That may have been the first time that I heard the word 'protocol,' and I guess I would have to say that was probably really the beginning, even though I had worked on this message passing system and I think that really was, by the way, the forerunner of many, many message passing systems we've seen since 1972. It's probably the beginning of my work in the real data network environment. These two other people, we went off to a corner of the mill in Maynard, and people who know the mill in Maynard, we managed to find a room that, I think, hadn't been opened up for hundreds of years and -- typical mill of Maynard; spiders on the ceiling, but we did find some corner. That was one nice thing about the mill is that you could always find yourself a corner hideaway somewhere and nobody could, probably, find you for weeks. That was a fascinating building, or set of buildings. It's sort of a network in its own right. Anyway, we went off, and these people spent about the first two weeks educating me. They educated me to this thing called bisync; binary synchronous communications, which I had never known about, and the concepts of message numbering and acknowledgments and transparency with escape characters, and all those wonderful things that bisync had. It was also just at that time that IBM -- I'm not sure whether or not they had released the very first versions of SDLC, but even if they hadn't, they had been working with the standards committee, and ANSI standards committee in the US, on a standard called ADCCP, Advanced Data Communications Control Protocol, ADCCP, which was going to be the US representative's input to the international standards committee for their protocol called HDLC, High-Level Data Link Control. At that time, the actual version that was happening at that time, what was called Single Numbered ADCCP, or Single Numbered HDLC, which to those of us in the data communications environment now thing of this as very bizarre -- it was a concept, but there was only one message number. There was not a message number independently in each direction, so that you would initialize the channel and then Station A would send his first message with number 1, Station B would then respond. Now, either he could respond with his data message, which would have number 2 in it, or just an acknowledgement, which would itself became message 2. The other one would then respond with message 3. There was only one number going back and forth. Now a days, when we think of it, it seems very bizarre to us, how you could operate in such an environment, but, let's face it, lots of things in the world of technology, once you move ahead and you understand the next step, things you had done previously crudely always seem strange. In fact, it would seem to us now - it seems sort of a trivial effort, for example, to design the wheel, which must have taken the cave man a long time to come to, but yet, once you have it, it obviously seems so obvious. So things just seem so obvious later. When you were back in that era, things weren't so obvious. It took you a long time to get there. They had started off on this one-numbered scheme, and I guess maybe they were talking about the potential concept of what eventually became known as dual numbering, but I think eventually the term dual numbering went away, and we don't use it anymore, but we eventually moved from single numbering to dual numbering. It was interesting, somewhere in that time frame, I actually attended one standards committee meeting. It was my one and only attendance at a standards committee meeting. It was held in the middle of the winter somewhere in a deserted beach in Virginia. You could get a real good deal on Holiday Inn rooms in those days; nobody went to vacation in the winter. So that was my only meeting, only because it was just -- it was taking up 80% of its time with parliamentary procedure, and who was going to make the next revision and how that was going to happen, and I guess I was just too much of a researcher and a doer to spend my time sitting in standards committees meetings, but I just went to one meeting and that was about it. Some of the people, though, that were there, even in those days -- 1974 -- as I understand are still on the committees today. IBM's representative at that committee was a guy named Bud Emmons, and Bud is still IBM's representative on the standards committees today. He's made a standard life of this effort. Anyway, I read what there was on the first version of ADCCP and on the first version I said, I believe, even the first version of SDLC may have been out, but at least what was in that was the concept of zero-bit insertion, or what is known as bit stuffing, today; the concept that the way that you would achieve transparency through this protocol was that every time you had five one-bits in a row, you would add a zero-bit.

Pelkey: The other way around.

Wecker: No. Is it other way around? No, zero-bit insertion. Right, if you had five ones in a row, you added a zero-bit, and that what was sitting on the two ends of the messages were these things called flags, which was six ones and a zero. So what you would do is you would make sure that it was not possible to have a flag imbedded in the middle of the message; you would make it unique. That concept, unfortunately, required all that to be done in hardware. We were off trying to design a protocol that could be used between DEC-10 and the PDP-11, and I think two significant things happened there -- well, I guess one real primary thing. The first was that -- and I'm not sure if I came up with the concept or if one of the other people did -- but what came out of our little three-man group was the concept that we should divorce control of the communication line from control of the device that we were managing on the end of the communication line. Previous protocols had actually intermixed those two concepts, in that acknowledgments that used to be sent back in previous protocols would not only acknowledge correct receipts of messages, but would also acknowledge, simultaneously, the proper processing of that message by the line printer or the card reader or the paper tape puncher, whatever strange devices we had on the end, and that the concept of managing the device at the end of the com line and managing the com line itself were sort of interwoven. What we said we were going to do is we said: "No, we're going to separate those two concepts. We're going to have one level call 'device control' -- called 'data link control.' We're going to manage the data link, and that above that, we're going to implement a level called 'device control.'" So I guess here was the beginning of our first two-layer architecture, so to speak, and I guess even below the data link was some kind of a physical level, so I guess we had a three-layer architecture, to be honest. We said we were going to divorce those two concepts from each other, and then the first thing we did is went off to concentrate on the data link control. I don't want to sound too conceited, but I guess I'm both a fast learner, and come up with some good ideas, and so, in a matter of a couple of weeks, I said: "First, I'm not really interested in inventing new hardware at DEC." In fact, our goal was to try to get this implemented as rapidly as possible so that we're going to use existing hardware which was, at that time, very character oriented, which is what all the bisync protocols ran on. So I said: "I want my protocol to continue to be character oriented, but I still want to achieve transparency in a better way than bisync achieved through the use of these DLE characters and DLESTX's and DLEETX's." Basically, if you think about it, what bisync did, and what the zero-bit insertion thing did was the same thing. In one case, they actually inserted entire characters called these DLE characters in order to achieve transparency. In zero-bit insertion, they were able to take that down to inserting only a single bit in order to achieve that transparency, but basically, you had to have some way -- it's the old quoting system, as I put it. It's a problem of 'how do you quote a quote inside a quoted sentence' and not interpret that quote as the ending quote of the sentence, but just a quote inside the sentence, and that's exactly the problem. The way we solve it in the English language is with a similar system of using double-quotes, and that's the concept of using a DLEDLE in the middle of the message to mean a data DLE, and that a DLESTX and ETX would frame it, and that way we would achieve the transparency we were looking for in the bisync case. I didn't like that kind of a scheme. It took a lot of software to do that processing, and instead I said: "I will use a count field. I will create a data link protocol based on the concept of a count field. I'm going to have field in the header that's going to simply say how long the data field is," and so you know longer have to search for a terminating character, and so there's no more need for DLE's and ETXs and STXs. In addition, I'm going to have a fixed-length header so I know exactly how long it is to begin with, and so solve the problem of a bit error occurring in the count field itself, which could clearly cause you, potentially, to go forever looking for the end of the message, is I put a block check in after the header, so the header would have its own block check field. The data would have its own block check field. You would check the header block check, which would validate your count field. You could then use that count field, proceed on, and read the rest of the message. It seemed simple enough, and we went with it. In addition, we went with the concepts of dual numbering -- that is separate numbers in each direction -- separate acknowledgments, and most of the other concepts that ADCCP had already adopted, which were things like piggy-backed acknowledgments, so you could acknowledge messages in the reverse direction, and this protocol eventually came out. We spent, I think, only seven weeks in this room, including my education process, and this was known as DDCMP, [Digital Data Communications Message Protocol]. It actually has been widely adopted by a number of smaller vendors as a data link protocol because it was so easy to implement, did not require special hardware. I was flattered to find it used in four or five university courses on data communication protocol design as a student exercise, and as I heard at one point, there were as many as 70 or 80 implementations, none of them used to communicate with DEC computers, just a protocol used in its own right. I believe that, had I probably been not so upset with standards committee meetings, and had spent much more time in the committees and had proposed this, there probably is a very good chance that his protocol could have overtaken IBM's effort at those meetings, and this could have been the base, instead, for HDLC, but IBM made the only proposals to the HDLC committee. No one else brought in any new ideas, and so HDLC basically turned out to be more or less a mirror image of IBM's SDLC protocol. They went on further, after IBM finished their effort, to then take their protocol and add more to it, including concepts that DDCMP had from day one, which HDLC did not adopt until many years later. HDLC was based the concept of a primary and a secondary station, or basically the concept of a master and a slave, in that it was the first version was meant to operate in a polled environment . . .

Tape Side Ends

Wecker: . . . called 'link management,' so there are really three parts to the DDCMP protocol: there was one, the framing portion, how do you get blocks across and know where the beginning and the end of the message was, that's where the count field came in; there was the message exchange portion, which had to do with positive and negative acknowledgements and time-outs and retransmissions, and that part was very independent of the count field part; and there was a third part which called 'link management,' which was 'who get's to transmit next on the channel?', or what we sometimes think of in local area networks as arbitration, which is where we come out with the differences between the Ethernet and the Token Ring. In the DDCMP protocol, we again divorced the concept of link management, which was the concept of 'when you could send,' from the concept of just exchanging messages, and so you could put the protocol into a primary/secondary environment on a polled line where the secondary stations then knew that they could not send until they were given permission via a polling message, but what they sent, once they got permission, was exactly what they would have sent on a full-duplex point-to-point line, where you always had permission and never needed it. So finally the concept that came out in HDLC many years later, called 'balanced mode,' which is what was finally used in the X.25 standardization effort, where the way they finally implemented in their mind was to have a primary and secondary station at both ends of the link, and that's the way they thought about it. DDCMP had that concept from day one. In fact, DDCMP was so symmetric that if you put a turn around plug on the end of the RS-232 line and you started DDCMP up, it would actually start up to itself. It was the only protocol that I know of that is that symmetric. It's actual initialization messages, you could send one, receive it, you would think the other end was sending it to you, you would send a message, you would get it back, you would send an acknowledge, you'd get the acknowledge back, and you would be merrily, happily talking to yourself and not knowing it. In fact, it was because of that, believe it or not, that we eventually, when the DECnet architecture got designed, we eventually had to put into the routing layer an identification message so that you would know that you were actually looped back to yourself, because if you put a plug at the end of a node, it would run the data link protocol all by itself, which, by the way, was a great testing feature, because you wouldn't even have to change one line of code in the protocol to do this. You wouldn't have to put it into a test mode. There was no test mode. You put a loop-back plug on, and it would run to itself. It was an interesting protocol. Anyway, we did the protocol. We also then did the set of device controlling protocols, specifically for the PDP-10 to a remote job entry station environment, which were controlling line printers and terminals. They were a bunch of application protocols, basically. That effort finished --

Pelkey: When did you complete that effort?

Wecker: Specs came out the door in '74 -- the specs came out the door. At that time, I guess I was really into networks and I thought about this protocol, and I said: "Gee, this DDCMP thing is pretty simple. I can implement this," so I went back to my message-pass operating system, the one I had done the previous year, and I implemented a DDCMP device driver, and I took the little messages that my kernel would have normally passed between the processes that -- or I used memory to just store the messages -- I now built this little program that would take the messages, pass them across a DDCMP synchronous link to another one of my machines, which would receive it, and pass it off to processes on the other machine. To keep life simple, I didn't have -- I used a very simple . . .

Interruption in the Interview

Wecker: . . . that is, I built this, quote, data link driver process, which could move the messages. To keep the addressing very simple, I simply named my processes starting off with the letter A for all processes on machine A, and B for all processes on machine B, and C for all processes on machine C, and so my routing process was really very simple. The little message passing kernel, when it got a message destined for a process name of B, and we were on machine A, would simply queue it down to this data link process. It would simply pass it over my synchronous communication channel at, in those days, the massive speed of 9.6 kilobits per second -- that's all we had -- and that machine would pick it up and would then pass it off to that process B -- correct process in that machine. I implemented the protocol, I tested it, it took a month or so, and just as with the multi-processor version, the network came up and ran. Nothing else had to be changed within the kernel. It was already a message pass system. I had built my first computer network. I had three PDP-11s connected in a triangular fashion, with 9.6 kilobit per second links; I had processes running in all of them. I could take a program that was on machine A, which used the right to A's line printer by sending things to the A line printer driver, I simply told it to send things to the B line printer driver. It made that one change to its process name, and stuff came out on the B system's line printer. I had built; actually, a very early distributed operating system. Again, there are many systems built today that are very similar. Processes had no idea where other processes were located; they simply were passing messages back and forth to each other. I actually built some kind of a small computer game. It wasn't chess, it wasn't checkers, it was some other computer game that you could run across three or four different computers, and they had to communicate with each other and pass each other various information, and I had that up and running and I actually think I demo'ed it at one of the DEC DECIS Conferences as an R&D effort, as just an R&D project. At that time—

Pelkey: This is '75, now--

Wecker: We're now into the beginnings of '75, I think. At that time, a number of the DEC product lines started getting requests from their customers to link DEC computers together; primarily the industrial product line and the laboratory data products line. One of the specific projects in the laboratory area was for Abbott Laboratories in Chicago, where they had experiments going on in the lab using PDP-11s for real-time process control, but they wanted to take the data that came off those process control studies, send it back to a DEC System 10 for analysis. Up to that point, people generally did that by carting around DEC tapes or mag tapes or magnetic cartridges. A network that we all know well, and a network that I refer to today as Sneaker Net. Today, you would do it with PCs and floppy disks, where you run across your hall in your sneakers, but in those days we used to use little DEC tapes. They wanted to get these machines hooked up more in a real-time fashion, and are able to pass that information back and forth. They also wanted to be able to pass information the other way. After they did the analysis, they wanted the DEC System 10 to be able to tweak some of the parameters of the real-time experiment. Other customers included Caterpillar Tractor, an early customer in the industrial products area, which was looking for ways to automate the assembly lines. They had a number of PDP-11s on their assembly lines that were used to control the assembly process. Again, they wanted to link these machines together so that they can pass control information to each other. Gordon Bell started passing my name around to the industrial and laboratory products groups as, quote, our guru on networking at DEC, and I started talking to their customers. I started talking to the industrial customers and the lab customers and starting to get requirements for what they were looking for. We set up a committee at DEC; everything, generally, at DEC, is done by committee. We set up a small network committee, which I, I guess, sort of headed up, and started thinking about what these requirements were and what we ought to do. Finally, I guess in '75, it got to be enough of an issue that Gordon decided that it was time that DEC finally move into this area and try to solve some of these networking problems for its customers. So I left the research group -- this was still early '75. Remember, everything we did at DEC didn't take too long. Things only took weeks, not years. Left the research group and went off to what was called Central Engineering, and as “Network Architect,” Relocatable to design a networking structure that could satisfy the needs of the industrial and laboratory products groups. I worked in the matrix organization. No one reported directly to me, but I had a group of four other individuals, one from each of the operating system groups that was assigned to actually implement this code on their particular computer, so there was someone from the PDP- 8, there was someone from the PDP-11 under RSX-11-M, and someone from the PDP-11 under RSX-11-D, which is a real-time system that is no longer in production, and there was someone there from the -- they may have been a representative from the DEC System 10 group as well. Central Engineering was really PDP-11s. The DEC 10 group was off, really, by itself in Marlboro at the time. The charter was for us to go off and to design a networking structure that DEC would be able to implement, to satisfy the needs of these laboratory and industrial customers, and I guess this was the beginning of the design of DECnet. Let me get something . . .

Interruption in the Interview

Wecker: . . . you will probably need this in front of you to help you understand it. So when we started the project, I knew very little about networking, and I think this is where DECnet starts to tie into other efforts that were happening at the time. I was not a member of the Arpanet community. I had read bits and pieces of things that were going on in Arpanet, and I guess Arpanet had been up and running at that time, I said my first step was to -- as a good researcher always does, which is what I was -- I was going to get up to speed on what computer networking was all about. I read various articles and papers, but I think there were two key texts that really I view as sort of the basis for DECnet, and let me also say that I have never been a believer of reinventing the wheel. I'm always a believer in building on what has been done before, which unfortunately is not always the way with a lot of computer people. The two most valuable pieces of information; [Standing and removing a book] the first thing is what I call my bible, which is this book, by Barber and Davies. You may have never seen this book. This book had essentially been reprinted some number of times. I'll show you how many times it's been reprinted.

Pelkey: It was first printed in 1973.

Wecker: '73, right.

Pelkey: (unintelligible) John Wiley and it's by Donald Davies and Derek Barber, called *Communication Networks for Computers*.

Wecker: Right, and this is the book that came out of their work at the National Physical Laboratory in their research on how to design computer networks. This book now costs over $110. I actually just bought this copy because, having left my last company, I lost my entire library, and I decided I really wanted this on the shelf.

Pelkey: I can understand why.

Wecker: When I met Donald Davies, even some number of years later at a number of conferences, I told him how valuable this book was. I don't think you could find me in the halls of DEC at that time without this book under my arm. I must have read this book cover to cover at least four times, and I have to give this book credit for forming a lot of the concepts that are in DECnet today; the concepts of its layering, the concepts of its very symmetric structure, the concepts of basically peer-to-peer networking, probably the concept that has most differentiated DECnet from IBM's SNA for the longest time. Those concepts really came out of this book. I really learned all about networks out of this book.

Pelkey: That's fascinating.

Wecker: I'm not sure Donald Davies actually even remembers that I used that much of this book as my basis for DECnet, but I do have to give him credit for all of that.

Pelkey: That's fantastic. What was the second document?

Wecker: The second document was Bob Metcalfe's PhD thesis.

Pelkey: I could have guessed.

Wecker: I don't remember how I latched onto it, but I managed to latch onto this thesis that Bob wrote on what was wrong with the Arpanet, basically, and I decided these guys had done a lot of work, they had learned a lot of things, and I was going to make sure that I didn't do anything in my network that Bob Metcalfe said I shouldn't do in that book, in his thesis. So with Bob Metcalfe's thesis and with the Davies and Barber Communication Networks for Computers book, and a number of other research papers that were coming out at the time -- remember, this was the time when I think a real heavy batch of research was going on in networks, including work on the Cyclades network by Louis Pouzin and Hubert

Zimmerman in France -- read all of their papers. They sent me a pile of papers. This was the time when the first annual datacom symposia had actually started, and we would meet and people like Vint Cerf from the Arpanet group, and Pouzin from Cyclades network, and Donald Davies out of the National Physical Laboratory Network, and Carl Sunshine who had been working in the ARPA effort, and -- God, many, many names. People who were --

Pelkey: Was that first datacom conference in '75?

Wecker: '75, I think it was, yeah.

Pelkey: Where was it held, do you remember?

Wecker: God, I could look it up. I don't remember.

Pelkey: I haven't gone and collected those papers.

Wecker: I don't remember, but you can get the proceedings from all of them. I don't remember -- and we went and that's when we were, among other things, we were discussing routing algorithms and transport protocols, but I set off to do DECnet. I said I had read the two; I had read Metcalfe's thesis and I had read this book. I looked at what we wanted to do, and the first concept was a fall out from what we had done in this DDCMP data link protocol, which is to say that what you want to do is create, in the network, a strong foundation, a strong base, and basically what you wanted to create was a message passing base. These concepts came out of -- I said the two books, they came out of my work on my distributed operating system, they came out of my work on this data link protocol. What you wanted to build was a strong base where processes executing in these different machines could send messages to each other. Probably the only difference from my original message passing work was that my message passing work was more of a -- operated more of a datagram orientation. We decided that, probably, the right thing to create between processes was more of a stream orientation, or what eventually became known in the trade as a virtual circuit. The term DECnet used was 'a logical link.' We'd create a link between processes. The two processes then connected, essentially, on this virtual wire, would be able to pass a stream of messages back and forth to each other. Then on top of that, you would implement a set of device and application management protocols for control of terminals and file systems and electronic mail and all the other services you wanted to put on. So we divorced the two from each other.

Pelkey: Why did you move toward virtual circuits, as opposed to staying with the datagram?

Wecker: At that level, I guess we thought that most of the things you did at the application level were stream-oriented things. We thought about controlling terminals, we thought about accessing files, we thought about transferring files. It seemed to us that everything we thought about at that level was a stream of messages being passed back and forth between the two processes.

Pelkey: But it was virtual, because it would share the same wire many --

Wecker: Right, it works out that we did have -- we do have a datagram concepts. It's lower down in the architecture. It's at the routing level. There is a datagram -- low enough down, it is a datagram network, and in fact, that concept, I have to give credit to Louis Pouzin and the Cyclades network for their implementing the interface between the routing layer and the transport layer as a datagram interface, and then implementing the interface at the transport layer as a stream interface --

Pelkey: Which is exactly the opposite of X.25.

Wecker: Right. Well, X.25 hadn't happened yet.

Pelkey: Right, but X.25 became the other way, because it was PTT pushed, and they wanted to have virtual at the lower layer, the routing layer, because that's the way their system was.

Wecker: Yes, exactly. So I guess -- I said I give Louis credit for the datagram concept. I thought about the problem --

Pelkey: This issue of creating streams was really one of being able to manage, not the connection but the process, as opposed to always having to open up and say: "Oh, I have something else," you always left the process open, in essence . . .

Interruption in Interview

Wecker: . . . Yeah, so what we said here, basically, is that --

Pelkey: At the lower level, it still stayed (unintelligible) datagram, but then you put a --

Wecker: Right, so we get the advantages of the packet switching concept, being able to share the channels adaptively, but at the upper part of the network, what users see is what users have always wanted to see, which is a sequential stream of information, because most of the application oriented things are not single message oriented applications, they are streams of messages: a file transfer, a conversation from a terminal, they're all streams of information back and forth. You don't really want to burden the application down with having to worry about managing that stream --

Pelkey: (unintelligible)

Wecker: Well, and doing his own numbering and keeping track of all of that. So the first concept was to build a strong base on which, later on, you could put a lot of application protocols. What that resulted in, unfortunately, was that, in the very first implementation that came out of DEC --

Pelkey: Which was when?

Wecker: Which were in early '76, I guess, late '76, I think. We have dates in here, and we'll get to that. What happened was that the product appeared, from a customer's point of view, to be fairly weak -- weak meaning that at that time, we had a file transfer protocol implemented, but we had no terminal protocol implemented yet. There were other vendors, like MODCOMP and Hewlett Packard, that had network products on the market at that same time frame, that had very strong terminal features, but they had implemented their network not as a layered structure, but as an integrated set of functions with a very weak base. I had a very hard time trying to convince customers, in those days, that this was a wonderful base, more things will come out in the future, you just have to believe that we were doing all the right things, and that you, as a customer, could implement your first application protocols initially to manage your terminals until DEC did, but that it was the strong base that was important.

Wecker: Now you had made a comment before I turned the tape on that in this book, The Ultimate Entrepreneur, that you read, there's a point in which there are some concerns that this might be too complex to implement. Was that around this time frame?

Wecker: I guess the feeling was -- this was in 1975 and '6 -- I guess the feeling was that networks were very, very complex things. They were not well understood and DEC felt that they better understood, believe it or not, how to build computers and how to build operating systems, and that maybe networks were just too complex for them. I think the thing that made this project work I think were maybe some of the same things that made the IBM PC project work; it was that at that time that I was working on this with this small group of people, this was a project off in the corner of DEC, it was not part of DEC's long term strategy, it was a small project to satisfy the needs of a couple of DEC product lines that had a few customers that had needs to interconnect some set of computers, and that, not that it was an interim effort, that it was on the order of magnitude of writing a compiler for a computer system. It was that kind of importance within DEC. It was going to be another set of tools, another set of products that came out on the PDP-11 family of computers that would let you link them together into small networks. In those days, literally, we were talking of five, six, seven machines, and that that was going to be it.

Pelkey: And it was rooted in the customer. You had gone out to listen to customers.

Wecker: I had gone out, listened to the customers and what the customers were looking to do and what they wanted, and it was not really viewed as, quote, "this is the new DEC strategy for the future," which is sort of ironic because, if you look at IBM, IBM, from the late '60s had already, in the very early '70s, had already put together their strategy that networking and distributed systems were going to be the key to IBM's future, and that, when the SNA project started at IBM, it started with the internal corporate plan that this was going to be a major part of IBM's future. So, what's ironic is that networking, with what they finally implemented, didn't do a thing for IBM, or hardly anything, and yet on the DEC side, we implemented all of this stuff, yet it was never in DEC's plans that networks were going to be DEC's center of attraction and main strategy, and yet that's what -- it is networks that have propelled DEC to, really, where they are today in the marketplace.

Pelkey: Where did DECnet and Ethernet and the VAX architecture come together?

Wecker: Not until a later phase. I'm going to get to that. So what we did in the beginning was, as I said, I read these books, I came out with the concept of creating a strong base, and the concept was based on this process structuring. That is, what we would do is we would create a base that would, at the user interface, give the user this concept of what we called a logical link, or I said what we think of classically today as a virtual circuit. You would go through a connection process, you'd be able to send and receive messages full duplex, and then you'd be able to tear down the circuit. Underneath that, from having read these various books, the concept of packet switching, to me, seemed very natural. I came out of building a message-passed operating system. As I said, that's what differentiated, in those days, the people in the computing world, who always thought in terms of messages and discrete blocks of information, compared to the people in the telecom and the telephone company world, who always thought in terms of circuits and real circuit paths that got nailed up. So the concepts of packet switching and being able to route these little packets around just seemed so natural to us.

Pelkey: And you also had already come to the conclusion about separating the communications from the processes that sat on top of the communications, so that the idea of having a separate box, service, was already out there.

Wecker: It was all part of it. Now, in the early days, in 1974 and '5, the problem was a very simple problem. It was laboratory process control environments. We were talking about point-to- point connections, we were talking about three, four, five, six computers, and if you think about it, the other concept that came on very early in the whole effort was this concept of peer level communications. There was not going to be any computer in charge of this network. These were going to be equal sized machines that were going to carry on a conversation with each other for the purpose of implementing some kind of a distributed environment. So the concept of peer level communications, the concepts of equality, carried on through all levels of the architecture and all of the protocols that we built after that. When you finally got to the application level, there was definitely the concept of a -- whether you want to call it a server and clients, or whether you want to call it a master and a slave, or however you want to term it at the application level -- the two processes were talking to each other, one was always the, quote, requestor and the requested, or one was the server and one was the requester of the server. So when you finally got to the application level, that's where the protocols tended to get non-symmetric, but below that, everything, as much as possible, was always kept symmetric and always kept equal, and as I said, this is probably THE major single difference from IBM's SNA.

Interruption in the Interview

Wecker: So, I said, the networks were peer level, they were simple, and they were small. We had the concept of layering, so we said, at the bottom there's clearly a physical layer, and then there's a need to control this wire. Well, by chance, I already had a protocol for that purpose, DDCMP. A lot of people think that the DDCMP data link protocol came after DECnet. It actually came before DECnet. It actually came out of that little effort with the DEC 10 remote job entry station, but I had a data link protocol, I said: "Great, I'm going to use DDCMP. I've got that problem solved." In those days, I didn't think about routing in the very first version, so I guess I had a null routing layer, so to speak. Then I needed my classical transport layer, and in my first version, well I already had sequencing of messages on my physical wire, and since I didn't have routing, all I had to have at that layer was really just a connection protocol. I didn't worry about lost messages because I only had one single path. Then, on top of my connection protocol, which was the beginnings of our transport, then I built -- oh, and I did have flow control at that level. In my transport protocol, I did put in a windowing flow control mechanism. Then, above that I started to build my application protocols, and I had a rudimentary session . . .

Tape Side Ends

Wecker: . . . in the implementation on the PDP-11, when you did a connection to a process, if that process was not active, the code would actually start the process running, so you didn't even have to have the process running on the machine. Then, at the application level, we built the beginnings of not only a file transfer, but again, thinking about it, I said: "File transfer is not a base mechanism. File access is a base mechanism," so the first protocol we built was a file access protocol. File transfer, then, simply became a utility program that accessed all the records of the file, which is a little bit different from what other people sometimes do. They build file transfer protocols. We never built a file transfer protocol; we built a file access protocol. So the first version got designed. A small group went off on two operating systems -- RSX-11-M and RSX-11-D -- on the PDP-11, two real time systems, and the PDP-8, and went off and did the first implementations. The first version on the PDP-11, from being assigned to work in the group to the first implementation rolling off into what you would call 'field test,' including my having to read the bible books and learn all about networks, eight months.

Pelkey: My goodness.

Wecker: Designed the architecture, learned about networks, worked with the engineers, ironed out problems and issues, designed all the protocols and did the first implementation. Now there were a few things missing from this implementation.

Pelkey: And this was not a 40-hour week you guys were working.

Wecker: It's pretty simple and straightforward. In fact, I've given numerous talks at various locations of IBM, and some people who really support DECnet within the IBM organization and think that we've done just the right things. It was simple and straightforward. I kept the protocols simple. Some of the things I learned from Metcalfe were this whole concept of variable length bytes was not the way to do things, and Arpanet got into a lot of trouble with all of that. I was implementing, at that point, on the standard eight-bit byte. All my fields were eight-bit oriented or multiples of eight-bit oriented, very fixed length headers. I did have a concept of a variable length field where I would use the high order bit to extend the field, but they were really pretty simple and straightforward. Simple addressing, nothing very esoteric. A lot of it came out of the previous work, either that I had done on my distributed system or the work we had done on the data link protocol level. Remember, at this point, we hadn't done routing, and we hadn't done some of the things at transport, but we built the first working network. The problem with it was it wasn't what I would call very user installable. We didn't have much of a 'netgen' procedure. You would have to do in and almost hand-build the tables that identified the physical data links and various parameters, and so it wasn't very easy. We shipped it to the first couple of customers, the customers where the DEC engineers went out and installed it in the field; it came up and ran fine. In fact, Caterpillar Tractor was a runner of Version 1 of DECnet.

Pelkey: Did Abbott put it up?

Wecker: Abbott was another story, which I'll get to in a second. The name DECnet itself, it's sort of interesting, we started out -- I guess DNA came out fairly early. IBM had done SNA and DNA seemed fairly obvious. In those days I used to refer to it as DNA, the basic building block of networks, sort of like the protein. Later on we changed it; I used to call it the Revised Network Architecture, RNA. It was sort of an inside joke. We originally called DECnet DEMOS, which stands for something like Digital Equipment Multi-Operating System Support, or something, after one of the Greek gods, and even though

DECnet seemed like the obvious name, DECnet was the name of the DEC internal mail system at that time. It was called DECnet. I think it was something they had purchased from some vendor who made a mail system run on DEC 10s. Finally, DEC decided that DECnet was the right name for the product family, and they gave up using DECnet as the internal name of the mail system. I forget what they changed it to. They changed it to something else, and DECnet, therefore, stuck as the name of the product family.

Pelkey: Do you recall when that was?

Wecker: That was in the '75 time frame.

Pelkey: So it was in the same time frame?

Wecker: Yeah it was in the same time frame. We did the first implementation. As time went on . . .

Interruption in the Interview

Wecker: . . . not that kind of a book. The DEC book was. What happened after that was a set of evolutions, and the next evolution, basically, started to move DECnet into larger markets and larger systems. Actually, the first thing that happened --

Pelkey: Which happened in '79, but between --

Wecker: Right, what happened between '74 and '79 was DEC embarked on a new computer program, which was probably the single significant event, called the VAX. DECnet architecture actually happened before the VAX, which, looking at it now, is actually a wonderful organization for the time frame for things to happen, because when the VAX architecture happened in 1976, DECnet had already gone through its first design phase, networks seemed to be catching on, Gordon Bell decided networks were definitely not only here to stay but that DEC was probably going to be making a mark in networks, and so the VAX should consciously incorporate DECnet into its base software architecture. I think that was a good key decision at that point. So what happened was, internally, to the VMS operating system, networks were part of the base VMS structure, in that names of files not only always began with a node name -- the concept of a node name was early on in the VMS structure -- and that node names always preceded file names, they were always carried in the data structures, the whole file access protocol was actually implemented within the file system of VMS, so we finally had an operating system here where the DECnet implementation was truly transparent to the user. The user would simply refer to a file name that happened to have a node name as part of it, and the file would get accessed, no matter where it was. He would do the same file open and close and get and put he would normally do, and the file could be on any node of the network. We had true transparent networking all through the whole VMS system. Had that not happened, had DECnet obviously happened later, what probably would have happened is something more akin to what we did on the PDP-11 operating systems, which is what we did on RSX-11-M and RSX-11-D, when we first implemented Phase I in 1975, is we had a set of subroutine calls which you call network file open and close and get and put, or you would call 'network connection create' and 'transmit' and 'receive,' so you would have a set of routines that you could call from high-level languages, which would then get you off to the process in the operating system that managed the network, but it was really a separate set of calls. In VMS, all that was basically eliminated. It was all integrated within the operating system.

Pelkey: So the time delay between these two was taking the time to - -

Wecker: Bring it up on the VAX --

Pelkey: Bring it up under the VMS and up under the VAX, and secondly, the process of productizing DECnet caused it to have to be (unintelligible) --

Wecker: And we just had to wait for demand to start building in networks. We had some initial customers, I said, which we sent the product out, but then over time, demand finally caught up and they finally got to the point that customers wanted larger networks, and I think the single thing that happened when we moved into the '79 time frame was the implementation of routing. At that time, I looked at a number of routing algorithms, and the algorithm we settled on was actually the early Arpanet algorithm -- not the hot potato -- what do you want to call it. The early one is the 'spanning tree', so it's the one before the spanning tree. It's the least cost routing algorithm, but again, having learned the lessons from the ARPA network, which was that the Arpanet's implementation of the least cost routing algorithm was -- let me start over on that. It takes me a few seconds sometimes to think about the past. The Arpanet implemented what was called the 'shortest delay' routing algorithm. What they would do is they would calculate what the delay times were between nodes and to get through nodes, and they would then take that information and pass it around from node to node, and each node would then update it's routing table, and then would readjust the routing paths. The problem with that is that from the time that you measure the information, or you measure the delays in the network, to the time that you can finally get all of the tables updated, the delays have actually changed, and what you wind up with is oscillations in the network. The way I like to describe this is the problem of two parallel highways. One highway is bumper to bumper, the other highway is empty; everybody in the bumper-to-bumper highway sees that, the next exit, they all get off and switch to the other highway. That highway then becomes bumper to bumper and the first highway is empty. They tend to oscillate back and forth, because you can't do the measurement and make the change instantaneously. The Arpanet, over time, eventually readjusted that algorithm to not make the changes every time they would measure delay, but they would measure delay over some number of minutes, and then they would make changes if there were drastic changes in delay. We borrowed the base algorithm from the ARPA Network, but made the change that we would only use a route until that route actually failed. We would pick -- and we made a change from delay to using a factor that we called 'cost.' Cost was a factor that the network manager at the site would assign to each link, and we never measured delay, but we would let him assign cost, for example, inversely proportional to link speed, which would therefore use the highest speed links you had. You could assign costs to anything you like, and what we would do is we would add up the costs of all the paths and figure out the path of least cost, so we called it 'least cost routing,' and we would stay with that until we actually had a failure in that path, and then we would switch to an alternate path.

Pelkey: Was there any systems optimization, i.e., the first time you set the path up it might take the least cost, but it might be a short duration, you might require that path for a short period of time, and yet sometime later would require the path for a longer time and would be put on a more costly route?

Wecker: No, the path between two end nodes -- if you had two virtual circuits between two end nodes, they would both always follow the same path. It was probably one of the minor weaknesses with the architecture, that there was no segregation of interactive traffic from batch traffic. We couldn't have those go by separate routes. That actually happened to one of the strong points of the IBM SNA architecture, this concept of what they call 'explicit routing,' where you could set up separate paths that different user virtual routes would take. That concept didn't exist. It was a simple algorithm, it worked on the fact that each node would send it's current costs to get to all the destinations to its neighbors, its neighbors would update its tables, if the tables got changed, that would be reflected to their neighbors. This would keep propagating until your tables were no longer changed, and then the algorithm settled down. When links failed or links came up, again, the changes in the tables would again be propagated. I think the algorithm worked well. I think you can argue a lot about the fact that the spanning tree algorithm is a little more rapid to adaptation, but in those days of small memory sizes -- and you got to remember, we were back, in the very first implementations, believe it or not, we were still on 64K byte machines. As I tell people today, you can't buy chips that small anymore. I remember, in those days, we implemented a booting process where we could have memory only computers on the network. We would down-line load them, and I had to do the entire down-line load protocol in 128 bytes of ROM code. We were on very small machines. These were the early days of the Arpanet. Even the Arpanet, remember, in those days, was on the Honeywell 316s and 516s. These were 64K byte machines as well.

Pelkey: And originally, some of them were --

Wecker: These were small machines, and we couldn't afford large spanning tree routing tables and the kinds of things that we do today, so I think at the time, that was the right routing algorithm, and in fact, that routing algorithm is still what is in DECnet today in Phase IV, still executing in 1988. The big thing is that we put routing in, and what routing meant was that we increased the size of the address space of the network, and it also meant that the transport protocol had to be updated to now include true message numbering and acknowledgements. That is, the transport protocol could not rely, any more, on the sequencing coming from the data link protocol. The decision we also made at the routing layer, again based on experience from the ARPA network, which was a very valuable teacher, was the problem of deadlock, in that we said the routing layer is going to protect itself from deadlock, and it's going to do it by offering a datagram service. It's not going to guarantee delivery of packets in the network. If it gets into trouble by running out of buffers, it's going to throw messages away to free buffers up. It's not going to let it get into a deadlock situation where it has all its buffers filled and it can't move on any further. We also did implement simple congestion control algorithms, but again, looking at what was happening in the research world, most of the congestion algorithms just added more traffic to the network. They kept sending too many messages around. We implemented very simple ones that simply, at one level, would just cut off traffic from the node itself being put into the network -- that is the node turned into a forwarding machine and not into a generation node -- and if the buffer level got even higher than that, then it would just start throwing away incoming packets until the buffer level went down. These simple algorithms basically guaranteed us we wouldn't get into a deadlock situation, and I think in many networks responsible for the general robustness of the whole DECnet architecture and structure. You have to be very careful not to get into algorithms that, on paper, are too elegant, but you can't implement them. So we added routing, we beefed up the transport, and now we truly had, at this point, what you would now view as a seven-layer architecture, or I guess what we called in DECnet a six-layer architecture. The one layer that we were missing -- or that we had merged -- were the presentation services and application layers, but in reality, in many networks even today, those layers are merged. That is, it's within the file protocol that you do all the file translations, or within the terminal protocol where you do all the terminal translations, and it's still pretty rare today that presentation services is a true distinct layer in its own right. Session had become a true layer, and it had some authorization controls and fields built into it.

Pelkey: Now, were you impacted by what was happening in the ISO community at this point?

Wecker: No, I think it was almost the other way around. Now it's true, at this point, DEC was still not a big player in the standards organization, but I guess that I have to believe that a lot of things that happened in the ISO seven-layer model must have come from a DECnet base, or there must have been an influence.

Pelkey: (Negative).

Wecker: No, you don't think so? None of the people had read anything?

Pelkey: Not that I know about. By March of '78, the seven-layer model had come forward.

Wecker: Yeah, but you don't think they had read anything from our '74 and '75 stuff?

Pelkey: Nobody had said a thing.

Wecker: Nobody said anything? Ok. Oh, the one decision that had been made, by the way, early on was that all of these protocols would be published, the architecture would be published, and that even though DEC did not encourage openness, it was the corporate policy that DECnet was an open architecture, even though today, even, very few people know that fact. Everybody thinks DECnet is very closed and proprietary. I'll also point out that everything we did in the design was always based around the fact that we were interconnecting 36-bit DEC System 10s, 16-bit PDP-11s, 32- bit VAXs, and 12 bit PDP-8s, and on the 11 alone, running three different operating systems, so that DECnet, internally, if you look at it, is not in the least DEC oriented. There's nothing that's DECish inside DECnet. The protocols are as generic as they could possibly be.

Pelkey: Now, Ethernet --

Wecker: The world didn't know that, and it's sort of interesting, because if you look at what's happened with other architectures just after that time, Xerox, which had put its architecture together for the office systems, XNS, one of the conscious things they did was put a really big PR campaign on to publicize XNS and to push XNS into the world, even though today XNS doesn't mean very much because Xerox has gotten out of the business. There were a number of the LAN vendors early on that picked up XNS as their architecture, including Bridge Communications.

Pelkey: Exactly right.

Wecker: Where DEC had always had an open architecture but never pushed it, and again, it's one of those things. I think if DEC had made a much stronger push, either through private PR --

Pelkey: Why do you think they didn't?

Wecker: They just didn't seem interested in either joining the standards community, being part of that effort, or having other machines at that time connect to DEC machines. It's not that they were so much against it, but they just weren't -- they took sort of a neutral position. They published the specs. There were a couple of small companies that hooked things up even at the DDCMP data link level with some machine tools, but not much activity. It's not that the company wanted to keep it closed; they just took a neutral position, I think, pretty much.

Pelkey: Now, what was the wiring scheme in the lower level, the physical data link layer?

Wecker: The physical link was still basically point-to-point. Oh, I should go back; let me make a comment early on. Back at the DDCMP protocol level, one of the nice things about a byte count protocol, and not the zero bit insertion, was that it kept everything on eight-bit byte boundaries, which meant you could run DDCMP asynchronously, as well as synchronously, which is something you can not do with a zero bit insertion protocol. What grew up was a large quantity of asynchronous DDCMP and asynchronous DECnet. People in many environments, where the data rates were fairly low, would hook their VAXs together, or their VAXs to PDP-11s, over asynchronous communication ports. They wouldn't have to buy any hardware. They already had asynchronous terminal ports. You could simply reassign one of those asynchronous ports to being a network port, and DECnet would run asynchronously at 9.6 kilobits.

Pelkey: So just over an RS-232 port.

Wecker: Standard asynchronous RS-232. You could use a null modem if the machines were only a couple of hundred feet apart. You wouldn't need anything but reverse --

Pelkey: You could use an async modem if they were farther apart.

Wecker: Right, right. There was a lot of 1200 bit per second DECnet, and in fact even today, the first version of DECnet DOS, the version DEC put out on the IBM PC under MS-DOS, and the first version was simply async between the PC and the VAXs.

Pelkey: How about that. So then Ethernet came in to play --

Wecker: So the connections here in '79 were basically point-to- point, synchronous and asynchronous, higher speed and lower speed. We had some -- there were DEC interfaces that would let you go up to 56 kilobits in some cases. Some customers had actually built worldwide networks, and then the evolution continued. In '83, DEC finally announced Phase IV of the architecture. I think the big thing that came in the '83 architecture was support for Ethernet local area networks. Ethernet fit in extremely well, because the -- not only had the routing layer been datagram oriented, but the transport layer specifically was implemented, at that point, not assuming anything from the data link and the routing layers. It had its own levels of data integrity, did its own message sequencing, and therefore when the DDCMP point-to-point wiring layer was replaced by an Ethernet, the rest of the architecture continued to function perfectly, even though occasionally an Ethernet packet might be corrupted, the transport layer took care of that anyway by itself. So Ethernet dropped in as another data link without any problem into the architecture. It offered a high-speed connection and the routing layer continued as it was before, and you could then build networks that had multiple Ethernets, had routing nodes between them. The other big thing that happened at this time was support for X.25, but support for X.25 as a data link within the architecture, which is, by the way, the same way that IBM supports it within SNA. Even though X.25 offers a virtual circuit service, if you're a vendor of network architectures like DEC, and you think about where you want to put X.25 in your network, you have to put it below your routing layer so that X.25 can serve as a piece of a path in your network, not as the total end to end path. Again, the view taken by comm vendors is X.25 IS the path of the network. If you're taken by computer vendors, its X.25 could be a piece of the path of the network, a very different view, and therefore all that's really used is a single virtual circuit is created between pairs of nodes that connect through the X.25 network, and that virtual circuit serves as the equivalent of the DDCMP data link, point-to- point. So it sits at the data link layer along with Ethernet, along with DDCMP point-to-point: three parallel data links. Again, it fits in quite nicely that way. The address space was increased, and in fact the address space was increased to finally include the concept of what we think of in the telephone network as area codes, or area addressing. The addressing scheme was broken into a two-level hierarchy; areas, and then nodes within areas, and that let us -- that's a way to reduce the size of the routing tables within all of the nodes, so that you then have area routing nodes and you then have nodes that route within areas. That network could be built today up to 65,000 nodes, even though I don't think there are any networks quite that large, but DEC's own internal network, as I understand it, is over 25,000 nodes today. It's the largest DECnet. Not only is it the largest DECnet, it's one of the largest packet-switched networks anywhere. These were the four phases, and I think these dates are about right, so you can use this stuff. There were a bunch of goals. If we go back a little bit, when we designed the architecture, I think there were seven main goals in this architecture. One was that it should really look the same on every machine it was built on. It's true that, on some machines, the interface would be a little more transparent than on other machines, but you should be able to take an application program you had running on a PDP-11 and, assuming it was written in the right high-level language, you should be able to move it over to one of the other DEC machines and, with a few minor changes in the calls to the network, you should be able to make it run. They should be sort of generically compatible. The network should support a wide range of communication facilities. When the architecture was designed, there was no Ethernet, there was no X.25, and there were no satellite links. All of that came in later, and all that supported very well within the architecture. We've compared a lot of things within DECnet against very custom tailored approaches, and I think a lot of this has hopefully gone away in most networks over time, but people should be less concerned about a protocol header that has two extra bytes of information, compared to what it may take to make a call through the operating system and get across the system interface. That's much more costly than what you're even going to lose from a couple of bytes in a header, and our philosophy was always 'build enough flexibility in here, and that over time, links will get faster, computers will get cheaper and faster, and it will take care of those problems, but if you don't leave enough flexibility in the basic structure, you're going to get caught short later.' It should support lots of topologies. In fact, this is probably a big difference between DECnet and the Arpanet. This is another interesting point. In the Arpanet -- first of all, the Arpanet was very much built along the same structural lines that a communication carrier would build a network. The Arpanet really consists of two networks, because there's the internal transport network and the external host network. Hosts sit around the outside of the network; in the middle is this thing called the transport network, and the two really have nothing to do with each other. The subnet transports information for the host network. They have two levels -- two separate addressing schemes, the way they talk to each other, and the hosts really can't talk to things in the subnet. That's the way the telephone network is. On the outside we have these things called telephones. They all have addresses, called phone numbers. They talk to each other. In the middle of the network we have telephone switches, but a telephone can't talk to a switch. They're really two separate networks. Computer vendors take a very different view of the network and a very different view of the world. Their view of the world is there really is no distinction, and this is true of both IBM's SNA and of DECnet as well, in that you have a set of interconnected machines, and these machines basically serve both functions at the same time. They serve host functions and . . .

Tape Side Ends

Wecker: . . . to keep routing tables updated, and if you implemented that in all your machines, and if a machine could take a message in an forward it through itself going out on another link, and sometimes serve as the termination of messages and sometimes just serve as the forwarding part of the message, yes, then you could build a, quote, single level network, but see, in the original ARPA network, that's not how it was ever built.

Interrupt in the Interview

Pelkey: Most of the time, if you were going to implement a TCP/IP environment now, you would do it with an IP that has the routing algorithm and the tables --

Wecker: Now you probably would, but in the early days of the Arpanet, and the days we were building DECnet and we looked at the Arpanet, the Arpanet made a distinction, and the hosts implemented -- they just built IP headers and dropped them in, where the subnet actually implemented the IP protocol. In DECnet, we didn't separate that. We said there is really only one network, and so the Arpanet had all these concepts called hosts and IMPs, which were switches, and TIPs, which were both terminals concentrators and switches. You couldn't have, in the Arpanet, terminals sitting on hosts, because of the way they had built their structure in those days. Terminals had to be on --

Pelkey: No, terminals could be on hosts.

Wecker: Terminals could be on hosts, I guess.

Pelkey: Right, in fact, that became a problem, that's why they created the TIP, because they were tying up too many of the hosts.

Wecker: But a TIP was both an IMP and a terminal server, no?

Pelkey: No. It was just a terminal server. It happened to be an IMP that got modified to be a terminal server.

Wecker: Right, but it couldn't be a switch. I guess that's what it was. It couldn't be a switch, so you still had to put a switch behind it, or somewhere on the network.

Pelkey: Well, the switches would always be connected to the computers.

Wecker: Right, but if you had a computer that didn't have any terminals ports and you had an IMP there, there was no way to just put a terminal onto it. You had to put an actual TIP in front of it.

Pelkey: You could not connect a terminal to an IMP directly.

Wecker: Right, and so we said, in DECnet, we said: "This is not the kind of flexibility we need. We need flexibility. We need the ability to take any computer sitting anywhere in the network and what would determine what a machine was what functions it had on it." If you didn't put any applications on it, then it would serve the function of an IMP, but if you put applications on it, it would serve the functions of a host, and it could be an IMP at the same time, and it could be a TIP at the same time. It could be all these things at the same time, depending what you put on it. It was just a one-level network. I think that was also a subtle distinction here. As I said, many of DECnet's networks had no routing at all. They were just point-to-point between the hosts. You cannot connect -- I think, in those days; you could not connect -- even if you could physically connect -- two hosts together. I don't believe the protocols were such that -- you couldn't, because the host spoke a host to IMP protocol, which was a specific protocol between hosts and IMPs, right. So you couldn't wire two hosts together, because they didn't speak a symmetric protocol to each other.

Pelkey: Right. Actually, Roberts and Merrill did that experiment between Lincoln Labs and SDC in '65, and they came to the conclusion that you could do that, through the timesharing operating systems, i.e. treat one like a virtual terminal for the purposes of accessing the other machine, but you couldn't do it over the physical link.

Wecker: Right, the physical link wouldn't work.

Pelkey: No, it just wouldn't work, but then the whole concept of creating host-resident protocols for direct connection, no one thought about doing that because they said: "If the links aren't any good, the question becomes how do you connect machines?" They clearly wanted to do -- in fact, when the Arpanet was created, the fact that there was so much electronic mail, burst oriented terminal traffic, was a great surprise, because they really expected there to be distributed applications, i.e. file transfers.

Wecker: Right, and it never happened.

Pelkey: It never happened.

Wecker: It's interesting, DECnet, on the other hand, because it went heavily into process control and laboratory environments, file transfers became a very heavy use early on. In fact, E-mail from DEC only happened in the last version of DECnet. In many, many versions, there was no E-mail. DECnet became a very heavy file protocol usage, which is sort of interesting.

Pelkey: That is sort of interesting.

Wecker: Again, because of the markets it was in specifically. So there was this distinction in that we only had a one level hierarchy. You could hook two machines together over a point-to- point link. They could be communicating in the network. Tomorrow, you could separate those machines apart and put routing nodes between, and they wouldn't know the difference. You had to make zero changes to those nodes. You could start a node off serving both host functions and routing functions. When the routing traffic through that node got to be too heavy, you could move that machine out, put a routing node in front of it, and again, it wouldn't have to change anything. It wouldn't know the difference. So, you could split the functions when you needed to. You could keep them together when it was adequate in performance, and all the nodes in the networks were addressed equally. There's only one level of addressing within the network. Hosts and IMPs and TIPs are all addressed as nodes in DECnet. There's only one level of addressing, which also means that a switch, or what you would call a router, has to actually implement all of DECnet because network management is implemented as an application process. So, you have to be able to create virtual circuits to switches, which you could say, on one hand, puts a little more burden on them; on the other hand, with the cost of computers today, in 1988, it hardly matters. So I think it was a wise decision at the time, and again, our decisions always revolved around technology will catch up with us.

Pelkey: I regrettably have to run, although I am thoroughly enjoying this conversation.

Wecker: How much time do you have?

Pelkey: I'm supposed to have a meeting in downtown Boston at 6:00.

Wecker: Ok, we'll wrap up in a few minutes. Other main concepts were that the network should be highly available. There should be nothing centralized. This is probably another major departure from IBM's SNA. There is no central node that serves as the authorization manager for virtual circuit creation, which there is in SNA, called the SSCP, the System Services Control Point. In fact, DECnet operates more or less the way the telephone system operates, which is in a very distributed fashion. There is no node in the telephone network that's in charge of all calls. The sending node builds connection packets, gets them routed to the destination, where a connection happens, which means it's very hard, sometimes, to control authorization access. You have to set up various password tables in every node of the network. There is not central depository for that.

Pelkey: One of the things that have been interesting to me in this conversation is that since DECnet and DDCMP preceded the VAX architecture that this issue of symmetry, which became peer-to- peer, in terms of its conceptual evolution, was inherent before there was the VAX strategy.

Wecker: Yes, it was.

Pelkey: It all came back to this issue about the DDCMP was so elegant and simple, that it led its way to doing peer-to-peer.

Wecker: Yes, exactly. There were five design -- six design principles drove the whole architecture. Be highly distributed. All the algorithms are distributed. Use a layered structure. All nodes are addressed uniformly; we just talked about that. Always trade off flexibility against absolute performance; machines will get faster. Implement things at the highest level you can, because that's where you get flexibility. Be dynamic and adaptive; in DECnet you can add new nodes to the network, you don't have to do any sysgen. The nodes find out about each other. All the algorithms are as adaptive as you can get them, and use simple, stable algorithms. That is, things like congestion control, make sure that we took the simplest way out, which is, when you're running out of buffers, you have to just discard packets, or in the post office scheme, that would be you burn the mail, as I tell people. You get logical links, which are the user paths. They're flow controlled virtual circuits. There is a segmentation scheme that -- or what you would call packetization, but in DECnet they're called segments. There's flow control. Oh yeah, another interesting point I should mention, because I think it's fairly critical, is the thing you talk to when you create a path is called an object. One of the things that, I guess, did not impress me about the ARPA network when I was first reading some of its protocols, were what's called the connection protocol and the concept of socket numbers; that you first had to go through a process of finding out what a process's socket was, and then you would connect to that socket. In DECnet, you just connect to the name of the thing you want to talk to. There are no socket numbers. These socket numbers, so to speak, are virtual identifiers that are picket up dynamically when you create the path, sort of similar to the way they're done in X.25, where you pick a number out of an available table.

Pelkey: The socket implies that there are only so many points of entry into a process.

Wecker: Right, and there really aren't. In DECnet, a process can build as many connections as it wants. There are not limits set in advance. Now, the reasons sockets were created were for well-known things, like a file server, you didn't have to worry about what its name was. We solved that problem by, for standard objects, having a type number. That is, there would be a number that would say 'this is a standard number,' like number three would be the file server process on all computers in DECnet, so you could say: "Either connect me to a name of something," which you would use between user and applications, or you'd say: "Connect me to object type three," which would then translate in a local table into a process name. You still don't need socket numbers, and that eliminated a whole initial protocol to get the socket numbers. Logical links are real simple. [Leafing through some papers] These are the base services in the network, and this just talks about the way things are done in the terminal protocol. The other interesting thing is DECnet network management. This shows how DECnet compares to the ISO layers, and how the layers operate. You can call me with any questions.

Pelkey: I'll certainly look through this.

Wecker: And this talks about routing, that there's a datagram network, and there are two layers of routing, and how addressing is done in all the four layers. The network management is completely distributed. Again, it's very much the way network management is now done, or proposed, by the ISO committee, that every system has a process called either the Network Management Process, or, in DECnet terminology, the Network Management Listener, which has hooks into all the layers of the architecture -- I have another picture over there -- and then it communicates to all the other nodes through an application protocol so they can exchange network management information. You can then decide where you want to put one or more network control programs, which means you can centralize it in one place and have that one location gather information from all the nodes of the network, or you could divide your network into regions, and have each regional network control center poll its local group of nodes for information, or you can have a node controlled locally at its own terminal and never talk to other nodes. You get all those choices, and a user makes that decision. Again, I think the difference is that this is a vendor architecture and not a carrier architecture, and I sometimes tell people: "DECnet is not a network. DECnet is a set of building blocks, pieces and tools from which users configured DECNets." That is, I can talk about the GE DECnet, or the DuPont DECnet, or DEC's own DECnet, but I can't talk about DECnet as a network. It is a structure. I can, however, talk about the Arpanet. That's a network. I can talk about its physical locations, its link speeds, and all of that, but I can't do that in DECnet. I think DECNet's basic strengths are its peer-to-peer orientation, which I think has carried on through all these years, its strong transport foundation, high function distribution in network management and in routing, pretty strong network management capabilities, very flexible addressing, and the fact that it's pretty compatible across all the machines. Where it's weak; I think it's weak, believe it or not, in a lot of things that I think SNA is strong in, but I don't think they're some of the key elements. Only a single active link is used between two communicating nodes. If you have two links between two nodes, they don't get used. Only one link will get used. SNA has the concept of what's called the Transmission Group, the ability to split traffic across multiple paths between nodes. No session service quality. This is the concept of being able to route interactive traffic one way and batch traffic another way. That concept does not really exist within DECnet today. I think it may be coming. No name server concept. Basically, you address things to a node address and then an object within the node. There is not a unique object name addressing completely in the network. You sort of have to know where things are, at least by a node name. You don't have to know where the node is, but you've got to know what node it's on, where, in other networks, you can just name things by some larger name, and there's no central authorization control. In many cases, people think that's a plus. In other networks, you could think that's a minus. Where I think things are going, I think they're going to add many of the things I just said. I think they're going to add transmission groups. They're going to add a third level of addressing, which I think is where the standards committee is going anyway, to a network address, and area, and then a node. That will let you interconnect networks. They're definitely putting in a name server. That's already been announced. This was a page that I created before Phase V got announced. A name server will be a node that will map a name onto a node name and object, so you can have a global name space, and that will let you move things around the network and not have to change the user's code. Today, if you moved an object from one node to another, you would have to change the addressing in the programs that want to talk to that thing. More network based applications. DEC has mail now. I think they will do, maybe distributed databases will be the next big step, and maybe distributed transaction processing will be the next big step. More things to use the network. They are doing a parallel ISO protocol set to let them connect to other vendors' systems, but that will not replace DECnet. It will be a parallel stream, and more network management. DEC has announced a NetView-like facility that'll allow you to connect other system to this. I think DNA was a major success, designed and implemented by a very small group of individuals. I think we did set modest goals and objectives for the effort. We built on small machines in the beginning. We didn't build huge networks. I think we took a straightforward approach to the problem. We didn't reinvent technology. I'm proud to say that I built on the National Physical Laboratory network and the ARPA network and the Cyclades network, designed it back in 1974; it is still as state of the art as any set of protocols and architecture today. Things have been added to keep it current that have just expanded its capabilities, but it's still as current as it was. I have a couple of things here on the routing layer itself, and I just wanted two more pictures.

Pelkey: I wanted to add that. Thank you very, very much for your time. This has been incredibly helpful, and I look forward to going through this and to reading the interview.

Wecker: I will also point out just a couple of more things for you . . .

END OF INTERVIEW