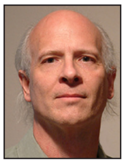
Interview with Jeff Mogul

RIK FARROW

Jeff Mogul works on fast, cheap, reliable, and flexible networking infrastructure for Google. Until 2013, he was a Fellow at HP Labs, doing research primarily on computer networks and operating systems issues for enterprise and cloud computer systems; previously, he worked at the DEC/Compaq Western Research Lab. He received his PhD from Stanford in 1986, an MS from Stanford in 1980, and an SB from MIT in 1979. He is an ACM Fellow. Jeff is the author or co-author of several Internet Standards; he contributed extensively to the HTTP/1.1 specification. He was an Associate Editor of *Internetworking: Research and Experience*, and has been the chair or co-chair of a variety of conferences and workshops, including SIGCOMM, OSDI, NSDI, HotOS, and ANCS.
jeffmogul@acm.org



Rik is the editor of *;login:*.
rik@usenix.org

I'm sure I met Jeff Mogul at a USENIX systems conference, but I can't remember which one. I had heard that Jeff was involved with the early Internet, but later than the groundbreaking work of Internet founders like Vint Cerf and Bob Kahn. And although I occasionally talked with Jeff, I knew little about him.

I did suspect he could shed some light on what it was like to manage an Internet connection in the mid-'80s and to help shape parts of TCP/IP. Jeff had also worked for Digital Equipment Corporation's (DEC) Western Research Lab. WRL was a small research lab in Palo Alto that produced a lot of pragmatic work and many papers too.

Rik Farrow: While at Stanford, you wrote several RFCs, including one about Reverse ARP, that allowed diskless workstations to learn their IP addresses, but also some early work on subnets. Can you tell us a little about how the Internet, and Stanford's Internets, appeared in 1984? I think that there are few people who know about early Ethernet and its limitations, as well as just how small (comparatively) the Internet was in those days.

Jeff Mogul: Actually, I think I had only a minor role in the RARP RFC. The subnet RFCs (RFCs 917, 919, 922, culminating in RFC 950) were more directly my work; I'm proud that Jon Postel co-authored that last one with me.

The Internet in 1984 was probably a lot like it was in 1983, at the time of the "TCP Transition"—I'm sure it had changed, but I don't remember what changed between 1983 and 1984. However, the TCP Transition was one of those events one remembers, because January 1, 1983 was the day that the predecessor to IP/TCP, called NCP, was disabled on the ARPANET, and so anyone who hadn't gotten TCP working by then would have been unable to send traffic [1].

At any rate, Stanford was connected to the ARPANET via Stanford's IMP; I think our IMP was number 11. IMPs had several ports, and so a few large computers could be connected to each IMP. I vaguely recall some kludges that were used to attach others. We also had an early "Experimental Ethernet" donated by Xerox PARC, along with a number of Xerox Alto computers. This Ethernet ran at 3 Mbps, and had 8-bit host addresses. Xerox had also developed a simple internetworking protocol, called PUP (PARC Universal Packet), which added an 8-bit network number, and I believe one could use Altos as routers between PUP networks. Bill Nowicki and I realized we could use some of the Stanford University Network, or "SUN," hardware (this was before Sun Microsystems was started) to build a really simple PUP router so that we didn't need to use precious Altos for that.

Once we realized that IP (and TCP) was coming, we needed a way to route IP packets from the ARPANET (effectively, the backbone of the future Internet) and the Stanford Ethernets. This meant installing an IP router at one of the IMP ports. I can't quite remember the chronology, but I do remember doing a lot of the work of installing and trying to set up this router. We used a PDP-11 for hardware, and I am pretty sure that we used J. Noel Chiappa's "C gateway" software; people then often used the term "gateway" instead of "router." I remember standing in our noisy machine room on lengthy long-distance phone calls to Noel (who was many time zones away) trying to debug his code in our router. That system was named GOLDEN-GATEWAY.STANFORD.EDU and had the address 10.1.0.11—Net 10 was the ARPANET; Stanford was IMP 11; the router was on port 1 of the IMP.

While fact-checking this, I found an old hosts.txt file [2] that included this line:

```
GATEWAY : 10.1.0.11, 36.40.0.62 : STANFORD-GATEWAY : LSI-11/23 :
MOS : IP/GW,GW/DUMB :
```

The MOS suggests that we were indeed using Noel's MIT router software.

At any rate, we also got something working by the TCP Transition date. I still have the button that Dan Lynch gave out, "I survived the TCP Transition." We also connected some of our BSD-based VAXes to the Ethernet via a card we got from Xerox, a driver we got from CMU, and some early IP/TCP software we got from BBN, the builders of the IMPs. I later took the CMU driver and generalized it in several ways. CMU had included a rudimentary packet filter in their driver, inspired by some Xerox Alto code, and I improved it enough to get an SOSP paper out of the deal [3]. Actually, I think we used the packet filter to implement PUP on the VAXes, so that might have happened before the TCP transition.

In those days, "RFC" really did stand for "Request for Comments"; pretty much anyone could write one and get a number assigned, without any actual review. The reason I wrote the original subnetting RFC was because the original "classful" IP addressing system allocated a single Class A network number to Stanford (36, or what we would call 36.0.0.0/8 once CIDR was invented). But we already had a bunch of Ethernets (18 according to RFC 917), so under this scheme we would have needed a lot more network numbers (one for each Ethernet), and we expected the number of Ethernets to grow. That would have bloated the Internet routing tables, still a problem today, even with CIDR. In those days, router memories were small—PDP11s had a 16-bit address space—and there wasn't a lot of spare bandwidth for exchanging routing updates, especially on the 56 Kbps ARPANET. Stanford was one of only a few Internet sites that actually had to worry about multiple subnets, which is why we had to invent the subnetting concept; I also wrote prototype code for BSD UNIX to implement this.

You asked about how small the Internet was in those days. It was definitely small in terms of backbone bandwidth (56 Kbps), the number of hosts (before DNS was invented, there was one Internet-wide "host table" file that we used to map names to addresses—I think SRI maintained and distributed it via FTP), and the number of people. There was a printed book that listed the name, address, phone, and email address of all known ARPANET users. And even in 1986 or 1987, people at academic networking conferences were still trying to figure out whether the Internet would ever be good for much of anything beyond email and FTP.

RF: Around 1987, you also wrote a technical report, and gave talks, about the harmfulness of fragmentation. Why had that become a problem?

JM: Internets can include different kinds of network technology, with different maximum packet sizes (so-called MTUs). Things are more homogeneous now than they were in the 1980s, when Ethernet hadn't quite taken over. At any rate, if you send a packet that fits within the MTU of the first-hop link, but some other link on the path has a smaller MTU, the router forwarding the packet at that point has to "fragment" the packet—divide it into smaller pieces that can be reassembled later. Several of us, including myself and also Chris Kent at Purdue (now Chris Kantarjiev) discovered a problem with fragmentation: sometimes it made TCP almost unusable. Why? Because our primitive Ethernet interfaces (NICs) could only buffer one or two received packets, so if packets arrived faster than the kernel could pull them out of the NIC buffer, some would get lost. This wasn't a huge problem for unfragmented packets, since the TCP receiver would get the first few packets and ACK them, and after a timeout, the sender would retransmit the rest: not ideal, but there was always forward progress.

However, when even one fragment of a fragmented packet is dropped, the receiver cannot reassemble the packet at all, so it is as if the whole packet were lost. To make matters worse, when the TCP sender eventually timed out and re-sent the packet, it would be fragmented again, and lost again with high probability, because these fragments generally arrived in bursts. So: no progress, and TCP users were sad.

This inspired Chris and me to publish a paper at SIGCOMM about the problem, and I led an IETF working group that (after a lot of debate) arrived at RFC 1191, defining "Path MTU Discovery"—which worked unless it didn't, and that's another long story that I mostly left for other people to solve.

RF: You worked on TCP, contributing the first open source firewall software, screend, to BSD UNIX. You later worked on the evolution of packet filtering in BSD, that lead to BPF. If I recall correctly, ULTRIX (DEC's UNIX) was based on BSD. Did DEC use screend as well?

JM: Screend and the packet filter were two mostly separate things. As I mentioned earlier, I think the original idea for packet filtering came from Xerox, but I think they used native code. Rich Rashid and Mike Accetta at CMU were inspired by that to add an interpreted packet filter to their Ethernet driver; interpretation (of a really simple instruction set) made it possible for user-mode programs to provide packet filters that could be safely interpreted within the kernel. I found it helpful to extend their filtering language in a variety of ways, and wrote the 1987 SOSP paper [3] describing this. But the so-called CMU-Stanford Packet Filter language was a rather inefficient stack-based execution model, and mostly one had to hand-code the filters. The Berkeley Packet Filter [4] replaced this with a register-style execution model, and they wrote a compiler for it, so overall it was much nicer, although I still think I had a cleaner solution for enabling programs such as tcpdump to put the Ethernet driver in "promiscuous mode" without having to make these programs setuid-root…but that's orthogonal to the interpreter design.

Screend came a few years later. Most of the BSD community gathered once or twice a year for a BSD summit meeting, and I believe we were at Berkeley for one of those the day that the Morris worm was unleashed. Bad timing! While that allowed a lot of people to focus on stopping the worm, they weren't able to install the patches needed.

Suddenly everyone realized that the original vision of the Internet as a place where any host could send any packet to any other host was actually not such a good one. The military had already realized this, and I think they installed "mail gateways" between the ARPANET and MILNET so that only email could get through; the rest of us thought that was rather typical of the military mind. So people started writing what we now call "firewalls."

I had already worked with Deborah Estrin (then of USC) and some of her grad students on a cryptographic approach of hers called "Visa protocols." With several decades of hindsight, you could call these "stateless SDN firewalls," since the Visa mechanisms used policy controllers separated from the routers. I believe our paper on this work was published after the Morris worm, but it was started earlier.

Anyway, at DEC in Palo Alto, Richard Johnsson (and perhaps others) needed to protect their computers against the Morris worm (and any copycats) *right now*, so he hacked a simple firewall into the BSD kernel. I think it either had a hard-coded ACL table, or perhaps there was a way to update it, but it wasn't very flexible or scalable. So I sat down and wrote screend, which did all of the fancy processing in user-mode code (in that respect, kind of like the packet-filter idea) and then kept a small cache of recent decisions in the kernel. It worked pretty well, I got a USENIX paper [5] out of the idea and helped DEC put it into the ULTRIX product, from which some colleagues ultimately built a (small) firewall business around it. I think my code even made it into the first setup for whitehouse.gov [6].

Yes, DEC's ULTRIX was very closely based on BSD, but of course with some DEC-specific additions, testing, documentation, etc.

RF: Right at the point where the Internet was growing exponentially, you worked on HTTP 1.1. What changes were you suggesting to improve the performance of HTTP around the mid-'90s?

JM: The original HTTP protocol would open a new TCP connection for each request, and then close it once the response was read. This turns out to make things really slow, because each request had to wait for the TCP handshake, which adds a network round trip. Network round-trip times (RTTs) are often tens or even hundreds of milliseconds and are the bane of good performance. Actually, it often added a lot more delay, because networks used to lose a lot more packets, and if your SYN was lost, your TCP had to time out and try again. Timeouts are usually much longer than RTTs. The other problem with the request-per-connection model was that each request-response transaction was serialized behind the previous one.

By making the TCP connections persistent [7], we avoided the setup costs. But we also enabled the use of "pipelining," a concept from computer architecture in which you can have several operations in flight at once. Since a typical Web page involves lots of HTTP requests (for images, CSS, etc.), once your browser downloads a page's HTML, it typically makes a large number of subsequent requests from the same server. With pipelining, the browser can launch a lot of those requests before any of the responses get back; this effectively allows us to hide all but one RTT.

Various things make persistent connections and pipelining harder to exploit in practice than we first realized; there are too many HTTP/1.1 servers that misbehave when asked to pipeline, so we had to wait for HTTP/2 before it became consistently safe to use. It took too long, but I think it proved to be a good idea.

RF: After you got your PhD from Stanford, you went to work for DEC's Western Research Lab in Palo Alto, California. What was it like to work in a research lab? Did you have total freedom to pick what you wanted to work on?

JM: WRL was an unusually wonderful environment. I don't think we ever had more than 25–30 researchers, and small number of other staff, but WRL people not only invented a lot of cool things at DEC, but many of them have become stars at other companies. I now work at Google, where many of our technical leaders started at WRL. Also, WRL hired people who were both talented and genuinely fun to work with—I have more friends from WRL than from any other era of my career.

WRL was even smaller when I joined, and it was just getting out of a narrow focus on building the first practical RISC computers, called Titans. In many ways, it was an academic environment—we hired people the same way that universities hire professors, we published papers, and we solved hard problems. However, we had more ability than universities to have a large group of people work on a single system, and we had the resources to build real hardware.

While many of us tended to look for our own problems to solve, within the context of the lab's mission (and we occasionally agonized over defining a mission statement), one would have to have been a fool not to remember that our nice salaries and offices were paid for by a profit-oriented business. WRL people wanted to change DEC (initially, by trying to convince DEC that RISC machines would be half as expensive as CISC machines), and so we tended to focus on solving problems that we thought the company needed to have solved. Sometimes we were willing to get ahead of DEC (as with RISC, and much later with AltaVista), but we realized that we needed to do things in a way that DEC could adopt without having to change lots of things at once. So, for example, we usually focused on C-based software, while our sister lab in Palo Alto (SRC, the Systems Research Center) focused on building clean-slate, top-to-bottom re-designs that promised much more wonderful results—but were really hard for DEC to absorb.

As a junior member of the lab, I was encouraged to spend some time following my own interests, but it was also made clear to me that I needed to commit substantial time to a project that contributed to the overall goals of the group. So, for example, my first major effort was to port the BSD networking stack into the Titan operating system, Tunix, a rather bizarre combination of some older BSD UNIX plus a lot of code written in Modula 2. Anita Borg, who joined WRL at about the same time, did her first major work on adding demand paging to Tunix.

RF: Any thoughts on the apparent decline of research labs, like WRL and Bell Labs?

JM: WRL declined rather suddenly. Compaq bought DEC in 1998 and absorbed the three existing research labs (WRL, SRC, and the Cambridge Research Lab) more or less intact, since Compaq had never had its own research organization. The Compaq experience had its good years, but by the end there just wasn't enough money to make things work, plus we were under some VPs who were not ideally suited to running a research organization. HP bought Compaq in 2002 and incorporated WRL, SRC, and CRL into HP Labs. Originally the idea was to keep our groups as separate parts of HP Labs, but that was unsustainable: while the DEC labs were fairly generalist, the other HP labs were very topic-focused, and the other lab directors apparently didn't like the idea of keeping our labs around. Shortly after that merger, WRL's director left to become an early Google employee, and after a few months of a rather uninspiring search for another director, HP dissolved WRL and moved us into the rest of the organization. To HP's credit, any WRL person who decided to leave at that time was compensated as if they had been laid off, and HP was still generous with layoff packages in 2002. SRC and CRL lasted somewhat longer.

I stayed at HP Labs for a decade, and for a while it was still a good place to do corporate research, but there were few upticks in a general decline. One person in particular did a lot of damage to the long-term prospects of HP Labs; that's a complex story, but I think the bottom line is that it is at best extremely hard to get value out of a corporate research lab these days, compared to simply waiting for a startup to invent what you need. The problem is that the typical reaction is to manage the research organization more intensely. ("You will innovate or else! And by the way, here are some stricter rules for how you will be creative.") I believe that's exactly backwards; I think Rick Rashid had it right when he said that as leader of Microsoft Research, he tried to ensure that they hired extremely carefully, and then he got out of the way. My view is that if you hire only researchers who are smart, who understand what the company needs, and who are internally motivated to make the company succeed, then a research lab has some chance of delivering value to the company, without micro-management from above.

But today, even that might not be enough for a corporate research lab to compete either with the massive number of startups or with companies like Google that integrate researchy people into product groups. And, in any case, once you've hired badly, you end up with an organization full of people who do not self-motivate in the right direction, and then you have to manage them aggressively, and from that you can never work your way back to a team of self-motivated, creative people.

The other big problem with corporate research labs is when the company's product groups aren't allowed to reserve some spare resources, for working collaboratively on tech transfer with researchers while the technology is still a bit risky. Tech transfer does still happen to those product groups, but typically it gets delayed until the group realizes it has to catch up with competitors. So the research result doesn't have an effect until it's too late to gain a real advantage from it. Researchers can still have a big impact on products by providing guidance and design reviews, but when the VP of Research only knows how to claim success for big-splash inventions, mere expertise-transfer isn't visible enough to get support or credit.

HP Labs is still hanging on (now in two separate companies, after HP split up), and there are still some smart people there, but with top people quitting every month, I don't think it will be interesting for much longer.

I have no direct experience with Bell Labs (or with AT&T Labs Research), so you should probably ask other people for those stories.

RF: In 2008, you were involved with a group talking about the future of system conferences. Did anything actionable come out of those discussions?

JM: Some discussions never end. I recently joined the NSDI Steering Committee, and we're currently in the middle of two different email threads about how to make systems conferences work better.

I suspect the discussions about the future of systems conferences started around five minutes into the first SOSP. That is, over 50 years ago. If two or more systems researchers are sitting in a bar, or going on a hike, or waiting for a bus, they will probably start discussing what is wrong with system conferences and how to fix them. For all I know, snake researchers also sit around moaning about the sorry state of herpetology conferences…but systems researchers are a bit weird in that, unlike almost all other scientific and engineering fields, we often put more emphasis on conference papers than journal papers, so we might be unusually interested in how conferences should be organized.

After joining more than my fair share of such BS sessions, and chairing a few conferences, I thought, "What better way to solve the problems of computer systems conferences and workshops than to have a workshop on that?" So I talked USENIX into letting me a run a workshop, WOWCS (Workshop on Organizing Workshops, Conferences, and Symposia for Computer Systems), co-located with NSDI '08, and we got a pretty nice selection of papers, plus a rousing discussion (which we wrote up as a *;login:* article in August 2008 [8].

People made some interesting proposals, but I haven't gone back over the material to see whether any of them bore fruit. There were a few papers on tools that have become indispensable (HotCRP and banal). Tom Anderson wrote a follow-up paper ("Conference Reviewing Considered Harmful" [9]) that presented some great data showing that PCs should not make their decisions based on reviewer scores; after that, when I've chaired PCs, I've warned people not to argue "we should take paper X over paper Y because it had a higher average score"—that's just amplifying some noise.

Since then, I participated in another workshop debating "publication culture in computing research" that wasted considerably more CO2, but I don't think it led to much change, either.

I think our emphasis in computer systems on conferences is the worst possible system…except for all of the other ones. In particular, I think when PC chairs pick well-intentioned PC members and run a face-to-face PC meeting carefully, the social structure of the meeting encourages reviewers to discuss papers with great passion and great integrity, because it's hard to hide bad or lazy behavior. I'm not sure how else to get that kind of combination.

References

[1] Flag day: https://www.internetsociety.org/blog/2013/01/30-years-tcp-and-ip-everything.

[2] Example of hosts.txt: https://emaillab.jp/pub/hosts/19840113/HOSTS.TXT.

[3] J. C. Mogul, R. F. Rashid, and M. J. Accetta, "The Packet Filter: An Efficient Mechanism for User-Level Network Code," in *Proceedings of the Eleventh ACM Symposium on Operating Systems Principles (SOSP)*, 1987, pp. 39–51.

[4] S. McCanne and V. Jacobson, "The BSD Packet Filter: A New Architecture for User-Level Packet Capture,"in *Proceedings of the Winter 1993 USENIX Annual Technical Conference*, pp. 259–269.

[5] J. C. Mogul, "Simple and Flexible Datagram Access Controls for Unix-Based Gateways," *in Proceedings of the Summer 1989 USENIX Technical Conference,* pp. 203–221.

[6] Section 1.2 mentions using screend as part of the firewall for whitehouse.gov: http://www.fwtk.org/fwtk/docs/documentation.html.

[7] J.C. Mogul. 1995. "The Case for Persistent-Connection HTTP," in *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM '95)*, pp. 299–313. See also http://www.hpl.hp.com/techreports/Compaq-DEC/WRL-95-4.pdf.

[8] J. C. Mogul (summarizer), "WOWCS '08: Workshop on Organizing Workshops, Conferences, and Symposia for Computer Systems," *;login:*, vol. 33, no. 4 (August 2008; online only): https://goo.gl/Gv5BPI.

[9] T. Anderson, "Conference Reviewing Considered Harmful," *SIGOPS Oper. Syst. Rev.,* vol. 43, no. 2 (April 2009), pp. 108–116. See also https://homes.cs.washington.edu/~tom/support/confreview.pdf.