Adaptive Threat Intelligence Framework for Proactive Cyber Defense

Emily Collins, PhD
Cybersecurity Institute for Advanced Research (CIAR)
ecollins@ciar.org

***Abstract*—In this paper, we present a novel approach for detecting advanced persistent threats (APTs) using deep learning techniques. APTs pose significant challenges to traditional security systems due to their stealthy and persistent nature. Our proposed method leverages a combination of convolutional neural networks and recurrent neural networks to analyze large-scale network traffic data. We introduce a novel attention mechanism that identifies subtle patterns in the data, enabling the detection of APTs with high accuracy. Experimental results on real-world datasets demonstrate the effectiveness of our approach in identifying previously unknown APTs while minimizing false positives. The framework offers a promising solution for enhancing the security posture of modern network infrastructures against sophisticated cyber threats.**

research!rsc:
Thoughts and links about programming, by Russ Cox: RSS

Computing History at Bell Labs

$$\alpha_{n,m_t} = \sum_{i,j=1}^{k} (A_3 A_1^{n_1-1} A_2 A_3 A_1^{n_2-1} A_2 \cdots A_3 A_1^{n_t-1} A_2)_{ij}.$$

Posted on Wednesday, April 9, 2008.

In 1997, on his retirement from Bell Labs, Doug McIlroy gave a fascinating talk about the "**History of Computing at Bell Labs**." Almost ten years ago I transcribed the audio but never did anything with it. The transcript is below.

My favorite parts of the talk are the description of the bi-quinary decimal relay calculator and the description of a team that spent over a year tracking down a race condition bug in a missile detector (reliability was king: today you'd just stamp "cannot reproduce" and send the report back). But the whole thing contains many fantastic stories. It's well worth the read or listen. I also like his recollection of programming using cards: "It's the kind of thing you can be nostalgic about, but it wasn't actually fun."

For more information, Bernard D. Holbrook and W. Stanley Brown's 1982 technical report "A History of Computing Research at Bell Laboratories (1937-1975)" covers the earlier history in more detail.

The authors would like to express their gratitude to the Cybersecurity Research Institute (CRI) for providing valuable resources and support during the research process

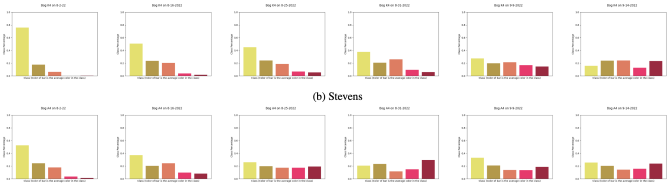
Corrections added August 19, 2009. Links updated May 16, 2018.

Update, December 19, 2020. The original audio files disappeared along with the rest of the Bell Labs site some time ago, but I discovered a saved copy on one of my computers: [MP3 | original RealAudio]. I also added a few corrections and notes from Doug McIlroy, dated 2015 [sic].

Transcript

Computing at Bell Labs is certainly an outgrowth of the mathematics department, which grew from that first hiring in 1897, G A Campbell. When Bell Labs was formally founded in 1925, what it had been was the engineering department of Western Electric. When it was formally founded in 1925, almost from the beginning there was a math department with Thornton Fry as the department head, and if you look at some of Fry's work, it turns out that he was fussing around in 1929 with trying to discover information theory. It didn't actually gel until twenty years later with Shannon.

1:10 Of course, most of the mathematics at that time was continuous. One was interested in analyzing circuits and propagation. And indeed, this is what led to the growth of computing in Bell Laboratories. The computations could not all be done symbolically. There were not closed form solutions. There was lots of numerical computation done. The math department had a fair stable of computers, which in those days meant people. [laughter]



(b) Stevens

(c) Crimson Queen

2:00 And in the late '30s, George Stibitz had an idea that some of the work that they were doing on hand calculators might be automated by using some of the equipment that the Bell System was installing in central offices, namely relay circuits. He went home, and on his kitchen table, he built out of relays a binary arithmetic circuit. He

decided that binary was really the right way to compute. However, when he finally came to build some equipment, he determined that binary to decimal conversion and decimal to binary conversion was a drag, and he didn't want to put it in the equipment, and so he finally built in 1939, a relay calculator that worked in decimal, and it worked in complex arithmetic. Do you have a hand calculator now that does complex arithmetic? Ten-digit, I believe, complex computations: add, subtract, multiply, and divide. The I/O equipment was teletypes, so essentially all the stuff to make such machines out of was there. Since the I/O was teletypes, it could be remotely accessed, and there were in fact four stations in the West Street Laboratories of Bell Labs. West Street is down on the left side of Manhattan. I had the good fortune to work there one summer, right next to a district where you're likely to get bowled over by rolling beeves hanging from racks or tumbling cabbages. The building is still there. It's called Westbeth Apartments. It's now an artist's colony.

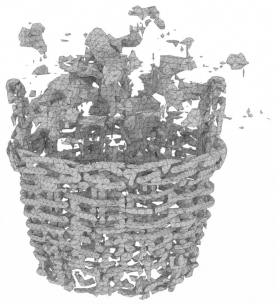
4:29 Anyway, in West Street, there were four separate remote stations from which the complex calculator could be accessed. It was not time sharing. You actually reserved your time on the machine, and only one of the four terminals worked at a time. In 1940, this machine was shown off to the world at the AMS annual convention, which happened to be held in Hanover at Dartmouth that year, and mathematicians could wonder at remote computing, doing computation on an electromechanical calculator at 300 miles away.

5:22 Stibitz went on from there to make a whole series of relay machines. Many of them were made for the government during the war. They were named, imaginatively, Mark I through Mark VI. I have read some of his patents. They're kind of fun. One is a patent on conditional transfer. [laughter] And how do you do a conditional transfer? Well these gadgets were, the relay calculator was run from your fingers, I mean the complex calculator. The later calculators, of course, if your fingers were a teletype, you could perfectly well feed a paper tape in, because that was standard practice. And these later machines were intended really to be run more from paper tape. And the conditional transfer was this: you had two teletypes, and there's a code that says "time to read from the other teletype". Loops were of course easy to do. You take paper and [laughter; presumably Doug curled a piece of paper to form a physical loop]. These machines never got to the point of having stored programs. But they got quite big. I saw, one of them was here in 1954, and I did see it, behind glass, and if you've ever seen these machines in the, there's one in the Franklin Institute in Philadelphia, and there's one in the Science Museum in San Jose, you know these machines that drop balls that go wandering sliding around and turning battle wheels and ringing bells and who knows what. It kind of looked like that. It was a very quiet room, with just a little clicking of relays, which is what a central office used to be like. It was the one air-conditioned room in Murray Hill, I think. This machine ran, the Mark VI, well I think that was the Mark V, the Mark VI actually went to Aberdeen. This machine ran for a good number of years, probably six, eight. And it is said that it never made an undetected error. [laughter]

8:30 What that means is that it never made an error that it did not diagnose itself and stop. Relay technology was very very defensive. The telephone switching system had to work. It was full of self-checking, and so were the calculators, so were the calculators that Stibitz made.

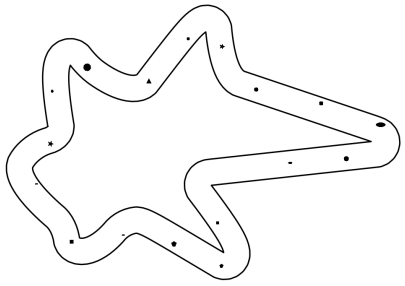
9:04 Arithmetic was done in bi-quinary, a two out of five representation for decimal integers, and if there weren't exactly two out of five relays activated it would stop. This machine ran unattended over the weekends. People would bring their tapes in, and the operator would paste everybody's tapes together. There was a beginning of job code on the tape and there was also a time indicator. If the machine ran out of time, it automatically stopped and went to the next job. If the machine caught itself in an error, it backed up to the current job and tried it again. They would load this machine on Friday night, and on Monday morning, all the tapes, all the entries would be available on output tapes.

Question: I take it they were using a different representation for loops and conditionals by then.



Doug: Loops were done actually by they would run back and forth across the tape now, on this machine.

$$M_{2t} = -iS'_{2t}(0) = -iD'_0(0)\mathbf{1} + D_0(0)M_t,$$



10:40 Then came the transistor in ’48. At Whippany, they actually had a transistorized computer, which was a respectable minicomputer, a box about this big, running in 1954, it ran from 1954 to 1956 solidly as a test run. The notion was that this computer might fly in an airplane. And during that two-year test run, one diode failed. In 1957, this machine called TRADIC, did in fact fly in an airplane, but to the best of my knowledge, that machine was a demonstration machine. It didn’t turn into a production machine. About that time, we started buying commercial machines. It’s wonderful to think about the set of different architectures that existed in that time. The first machine we got was called a CPC from IBM. And all it was was a big accounting machine with a very special plugboard on the side that provided an interpreter for doing ten-digit decimal arithmetic, including opcodes for the trig functions and square root.

$$\varphi_i f = -\frac{(1+\beta y_i)f - (1+\beta y_{i+1})f|_{y_i \leftrightarrow y_{i+1}}}{y_i - y_{i+1}},$$

12:30 It was also not a computer as we know it today, because it wasn’t stored program, it had twenty-four memory locations as I recall, and it took its program instead of from tapes, from cards. This was not a total advantage. A tape didn’t get into trouble if you dropped it on the floor. [laughter]. CPC, the operator would stand in front of it, and there, you would go through loops by taking cards out, it took human intervention, to take the cards out of the output of the card reader and put them in the ?top?. I actually ran some programs on the CPC ?. . . ?. It’s the kind of thing you can be nostalgic about, but it wasn’t actually fun. [laughter]

13:30 The next machine was an IBM 650, and here, this was a stored program, with the memory being on drum. There was no operating system for it. It came with a manual: this is what the machine does. And Michael Wolontis made an interpreter called the L1 interpreter for this machine, so you could actually program in, the manual told you how to program in binary, and L1 allowed you to give something like 10 for add and 9 for subtract, and program in decimal instead. And of course that machine required interesting optimization, because it was a nice thing if the next program step were stored somewhere – each program step had the address of the following step in it, and you would try to locate them around the drum so to minimize latency. So there were all kinds of optimizers around, but I don’t think Bell Labs made ?. . . ? based on this called “soap” from Carnegie Mellon. That machine didn’t last very long. Fortunately, a machine with core memory came out from IBM in about ’56, the 704. Bell Labs was a little slow in getting one, in ’58. Again, the machine came without an operating system. In fact, but it did have Fortran, which really changed the world. It suddenly made it easy to write programs. But the way Fortran came from IBM, it came with a thing called the Fortran Stop Book. This was a list of what happened, a diagnostic would execute the halt instruction, the operator would go read the panel lights and discover where the machine had stopped, you would then go look up in the stop book what that meant. Bell Labs, with George Mealy and Gwen Hanson, made an operating system, and one of the things they did was to bring the stop book to heel. They took the compiler, replaced all the stop instructions with jumps to somewhere, and allowed the program instead of stopping to go on to the next trial. By the time I arrived at Bell Labs in 1958, this thing was running nicely.

[*McIlroy comments, 2015*: I’m pretty sure I was wrong in saying Mealy and Hanson brought the stop book to heel. They built the OS, but I believe Dolores Leagus tamed Fortran. (Dolores was the most accurate programmer I ever knew. She’d write 2000 lines of code before testing a single line–and it would work.)]

$$w_t^\alpha = v_{t+1}^\alpha - v_t^\alpha, \qquad w_t^\beta = v_{t+1}^\beta - v_t^\beta.$$

16:36 Bell Labs continued to be a major player in operating systems. This was called BESYS. BE was the SHARE abbreviation for Bell Labs. Each company that belonged to Share, which was the IBM users group, ahd a two letter abbreviation. It’s hard to imagine taking all the computer users now and giving them a two-letter abbreviation. BESYS went through many generations, up to BESYS 5, I believe. Each one with innovations. IBM delivered a machine, the 7090, in 1960. This machine had interrupts in it, but IBM didn’t use them. But BESYS did. And that sent IBM back to the drawing board to make it work. [Laughter]

$$A^* = z \text{ and } A = \frac{\partial}{\partial z}$$

17:48 Rob Pike: It also didn't have memory protection.

Doug: It didn't have memory protection either, and a lot of people actually got IBM to put memory protection in the 7090, so that one could leave the operating system resident in the presence of a wild program, an idea that the PC didn't discover until, last year or something like that. [laughter]

Big players then, Dick Hamming, a name that I'm sure everybody knows, was sort of the numerical analysis guru, and a seer. He liked to make outrageous predictions. He predicted in 1960, that half of Bell Labs was going to be busy doing something with computers eventually. ?...? exaggerating some ?...? abstract in his thought. He was wrong. Half was a gross underestimate. Dick Hamming retired twenty years ago, and just this June he completed his full twenty years term in the Navy, which entitles him again to retire from the Naval Postgraduate Institute in Monterey. Stibitz, incidentally died, I think within the last year. He was doing medical instrumentation at Dartmouth essentially, near the end.

[*McIlroy comments, 2015*: I'm not sure what exact unintelligible words I uttered about Dick Hamming. When he predicted that half the Bell Labs budget would be related to computing in a decade, people scoffed in terms like "that's just Dick being himelf, exaggerating for effect".]

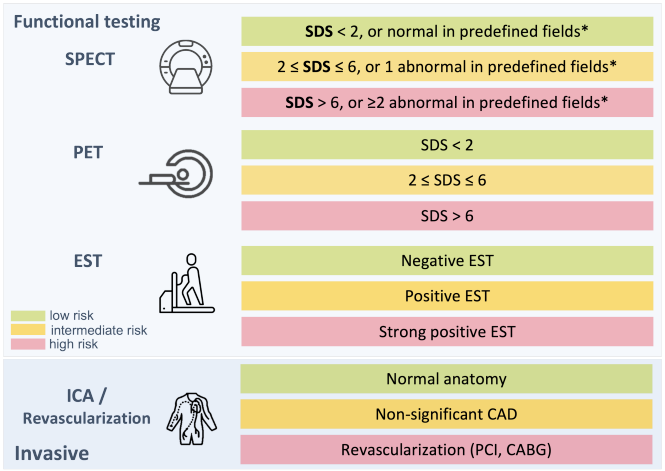
20:00 Various problems intrigued, besides the numerical problems, which in fact were stock in trade, and were the real justification for buying machines, until at least the '70s I would say. But some non-numerical problems had begun to tickle the palette of the math department. Even G A Campbell got interested in graph theory, the reason being he wanted to think of all the possible ways you could take the three wires and the various parts of the telephone and connect them together, and try permutations to see what you could do about reducing sidetone by putting things into the various parts of the circuit, and devised every possibly way of connecting the telephone up. And that was sort of the beginning of combinatorics at Bell Labs. John Reardon, a mathematician parlayed this into a major subject. Two problems which are now deemed as computing problems, have intrigued the math department for a very long time, and those are the minimum spanning tree problem, and the wonderfully ?comment about Joe Kruskal, laughter?

$$-2 = -3 - 2b + 3n = -2b + 3(n-1).$$

21:50 And in the 50s Bob Prim and Kruskal, who I don't think worked on the Labs at that point, invented algorithms for the minimum spanning tree. Somehow or other, computer scientists usually learn these algorithms, one of the two at least, as Dijkstra's algorithm, but he was a latecomer.

[*McIlroy comments, 2015*: I erred in attributing Dijkstra's algorithm to Prim and Kruskal. That honor belongs to yet a third member of the math department: Ed Moore. (Dijkstra's algorithm is for shortest path, not spanning tree.)]

Another pet was the traveling salesman. There's been a long list of people at Bell Labs who played with that: Shen Lin and Ron Graham and David Johnson and dozens more, oh and ?...?. And then another problem is the Steiner minimum spanning tree, where you're allowed to add points to the graph. Every one of these problems grew, actually had a justification in telephone billing. One jurisdiction or another would specify that the way you bill for a private line network was in one jurisdiction by the minimum spanning tree. In another jurisdiction, by the traveling salesman route. NP-completeness wasn't a word in the vocabulary of lawmakers [laughter]. And the Steiner problem came up because customers discovered they could beat the system by inventing offices in the middle of Tennessee that had nothing to do with their business, but they could put the office at a Steiner point and reduce their phone bill by adding to what the service that the Bell System had to give them. So all of these problems actually had some justification in billing besides the fun.



24:15 Come the 60s, we actually started to hire people for computing per se. I was perhaps the third person who was hired with a Ph.D. to help take care of the computers and I'm told that the then director and head of the math department, Hendrick Bode, had said to his people, "yeah, you can hire this guy, instead of a real mathematician, but what's he gonna be doing in five years?" [laughter]

25:02 Nevertheless, we started hiring for real in about '67. Computer science got split off from the math department. I had the good fortune to move into the office that I've been in ever since then. Computing began to make, get a personality of its own. One of the interesting people that came to Bell Labs for a while was Hao Wang. Is his name well known? [Pause] One nod. Hao Wang was a philosopher and logician, and we got a letter from him in England out of the blue saying "hey you know, can I come and use your computers? I have an idea about theorem proving." There was theorem proving in the air in the late 50s, and it was

mostly pretty thin stuff. Obvious that the methods being proposed wouldn't possibly do anything more difficult than solve tic-tac-toe problems by enumeration. Wang had a notion that he could mechanically prove theorems in the style of Whitehead and Russell's great treatise Principia Mathematica in the early patr of the century. He came here, learned how to program in machine language, and took all of Volume I of Principia Mathematica – if you've ever hefted Principia, well that's about all it's good for, it's a real good door stop. It's really big. But it's theorem after theorem after theorem in propositional calculus. Of course, there's a decision procedure for propositional calculus, but he was proving them more in the style of Whitehead and Russell. And when he finally got them all coded and put them into the computer, he proved the entire contents of this immense book in eight minutes. This was actually a neat accomplishment. Also that was the beginning of all the language theory. We hired people like Al Aho and Jeff Ullman, who probed around every possible model of grammars, syntax, and all of the things that are now in the standard undergraduate curriculum, were pretty well nailed down here, on syntax and finite state machines and so on were pretty well nailed down in the 60s. Speaking of finite state machines, in the 50s, both Mealy and Moore, who have two of the well-known models of finite state machines, were here.

$$\left\{x \in B_{\frac{R}{8}} : u < \mu_- + \frac{\omega}{2^{l+5}}\right\} \subset B_{\frac{R}{4}}.$$

28:40 During the 60s, we undertook an enormous development project in the guise of research, which was MULTICS, and it was the notion of MULTICS was computing was the public utility of the future. Machines were very expensive, and ?indeed? like you don't own your own electric generator, you rely on the power company to do generation for you, and it was seen that this was a good way to do computing – time sharing – and it was also recognized that shared data was a very good thing. MIT pioneered this and Bell Labs joined in on the MULTICS project, and this occupied five years of system programming effort, until Bell Labs pulled out, because it turned out that MULTICS was too ambitious for the hardware at the time, and also with 80 people on it was not exactly a research project. But, that led to various people who were on the project, in particular Ken Thompson – right there – to think about how to – Dennis Ritchie and Rudd Canaday were in on this too – to think about how you might make a pleasant operating system with a little less resources.

30:30 And Ken found – this is a story that's often been told, so I won't go into very much of unix – Ken found an old machine cast off in the corner, the PDP-7, and put up this little operating system on it, and we had immense GE635 available at the comp center at the time, and I remember as the department head, muscling in to use this little computer to be, to get to be Unix's first user, customer, because it was so much pleasanter to use this tiny machine than it was to use the big and capable machine in the comp center. And of course the rest of the story is known to everybody and has affected all college campuses in the country.

31:33 Along with the operating system work, there was a fair amount of language work done at Bell Labs. Often curious off-beat languages. One of my favorites was called Blodi, B L O D I, a block diagram compiler by Kelly and Vyssotsky. Perhaps the most interesting early uses of computers in the sense of being unexpected, were those that came from the acoustics research department, and what the Blodi compiler was invented in the acoustic research department for doing digital simulations of sample data system. DSPs are classic sample data systems, where instead of passing analog signals around, you pass around streams of numerical values. And Blodi allowed you to say here's a delay unit, here's an amplifier, here's an adder, the standard piece parts for a sample data system, and each one was described on a card, and with description of what it's wired to. It was then compiled into one enormous single straight line loop for one time step. Of course, you had to rearrange the code because some one part of the sample data system would feed another and produce really very efficient 7090 code for simulating sample data systems. By in large, from that time forth, the acoustic department stopped making hardware. It was much easier to do signal processing digitally than previous ways that had been analog. Blodi had an interesting property. It was the only

programming language I know where – this is not my original observation, Vyssotsky said – where you could take the deck of cards, throw it up the stairs, and pick them up at the bottom of the stairs, feed them into the computer again, and get the same program out. Blodi had two, aside from syntax diagnostics, it did have one diagnostic when it would fail to compile, and that was "somewhere in your system is a loop that consists of all delays or has no delays" and you can imagine how they handled that.

35:09 Another interesting programming language of the 60s was Ken Knowlten's Beflix. This was for making movies on something with resolution kind of comparable to 640x480, really coarse, and the programming notion in here was bugs. You put on your grid a bunch of bugs, and each bug carried along some data as baggage, and then you would do things like cellular automata operations. You could program it or you could kind of let it go by itself. If a red bug is next to a blue bug then it turns into a green bug on the following step and so on. 36:28 He and Lillian Schwartz made some interesting abstract movies at the time. It also did some interesting picture processing. One wonderful picture of a reclining nude, something about the size of that blackboard over there, all made of pixels about a half inch high each with a different little picture in it, picked out for their density, and so if you looked at it close up it consisted of pickaxes and candles and dogs, and if you looked at it far enough away, it was a reclining nude. That picture got a lot of play all around the country.

Lorinda Cherry: That was with Leon, wasn't it? That was with Leon Harmon.

Doug: Was that Harmon?

Lorinda: ?. . . ?

Doug: Harmon was also an interesting character. He did more things than pictures. I'm glad you reminded me of him. I had him written down here. Harmon was a guy who among other things did a block diagram compiler for writing a handwriting recognition program. I never did understand how his scheme worked, and in fact I guess it didn't work too well. [laughter] It didn't do any production ?things? but it was an absolutely immense sample data circuit for doing handwriting recognition. Harmon's most famous work was trying to estimate the information content in a face. And every one of these pictures which are a cliche now, that show a face digitized very coarsely, go back to Harmon's first psychological experiments, when he tried to find out how many bits of picture he needed to try to make a face recognizable. He went around and digitized about 256 faces from Bell Labs and did real psychological experiments asking which faces could be distinguished from other ones. I had the good fortune to have one of the most distinguishable faces, and consequently you'll find me in freshman psychology texts through no fault of my own.

39:15 Another thing going on the 60s was the halting beginning here of interactive computing. And again the credit has to go to the acoustics research department, for good and sufficient reason. They wanted to be able to feed signals into the machine, and look at them, and get them back out. They bought yet another weird architecture machine called the Packard Bell 250, where the memory elements were mercury delay lines.

Question: Packard Bell?

Doug: Packard Bell, same one that makes PCs today.

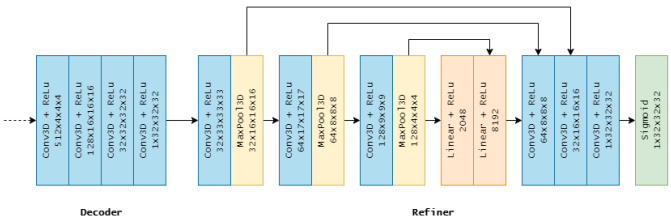
40:10 They hung this off of the comp center 7090 and put in a scheme for quickly shipping jobs into the job stream on the 7090. The Packard Bell was the real-time terminal that you could play with and repair stuff, ?. . . ? off the 7090, get it back, and then you could play it. From that grew some graphics machines also, built by ?. . . ? et al. And it was one of the old graphics machines in fact that Ken picked up to build Unix on.

$$\phi(\theta, \psi) = (\cos\theta \sin\psi, \sin\theta \sin\psi, \cos\psi)$$

40:55 Another thing that went on in the acoustics department was synthetic speech and music. Max Mathews, who was the the director of the department has long been interested in computer music. In fact since retirement he spent a lot of time with Pierre Boulez in Paris at a wonderful institute with lots of money simply for making synthetic music. He had a language called Music 5. Synthetic speech or, well first of all simply speech processing was pioneered particularly by John Kelly. I remember my first contact with speech processing. It was customary for computer operators, for the benefit of computer operators, to put a loudspeaker on the low bit of some register on the machine, and normally the operator would just hear kind of white noise. But if you got into a loop, suddenly the machine would scream, and this signal could be used to the operator "oh the machines in a loop. Go stop it and go on to the next job." I remember feeding them an Ackermann's function routine once. [laughter] They were right. It was a silly loop. But anyway. One day, the operators were ?. . . ?. The machine started singing. Out of the blue. "Help! I'm caught in a loop.". [laughter] And in a broad Texas accent, which was the recorded voice of John Kelly.

43:14 However. From there Kelly went on to do some speech synthesis. Of course there's been a lot more speech

synthesis work done since, by 43:31 folks like Cecil Coker, Joe Olive. But they produced a record, which unfortunately I can't play because records are not modern anymore. And everybody got one in the Bell Labs Record, which is a magazine, contained once a record from the acoustics department, with both speech and music and one very famous combination where the computer played and sang "A Bicycle Built For Two".



?...?

$$s(x) := |\{x_i : x_i = -\infty\}|.$$

44:32 At the same time as all this stuff is going on here, needless to say computing is going on in the rest of the Labs. it was about early 1960 when the math department lost its monopoly on computing machines and other people started buying them too, but for switching. The first experiments with switching computers were operational in around 1960. They were planned for several years prior to that; essentially as soon as the transistor was invented, the making of electronic rather than electromechanical switching machines was anticipated. Part of the saga of the switching machines is cheap memory. These machines had enormous memories – thousands of words. [laughter] And it was said that the present worth of each word of memory that programmers saved across the Bell System was something like eleven dollars, as I recall. And it was worthwhile to struggle to save some memory. Also, programs were permanent. You were going to load up the switching machine with switching program and that was going to run. You didn't change it every minute or two. And it would be cheaper to put it in read only memory than in core memory. And there was a whole series of wild read-only memories, both tried and built. The first experimental Essex System had a thing called the flying spot store which was large photographic plates with bits on them and CRTs projecting on the plates and you would detect underneath on the photodetector whether the bit was set or not. That was the program store of Essex. The program store of the first ESS systems consisted of twistors, which I actually am not sure I understand to this day, but they consist of iron wire with a copper wire wrapped around them and vice versa. There were also experiments with an IC type memory called the waffle iron. Then there was a period when magnetic bubbles were all the rage. As far as I know, although microelectronics made a lot of memory, most of the memory work at Bell Labs has not had much effect on ?...?. Nice tries though.

48:28 Another thing that folks began to work on was the application of (and of course, right from the start) computers to data processing. When you owned equipment scattered through every street in the country, and you have a hundred million customers, and you have bills for a hundred million transactions a day, there's really some big data processing going on. And indeed in the early 60s, AT&T was thinking of making its own data processing computers solely for billing. Somehow they pulled out of that, and gave all the technology to IBM, and one piece of that technology went into use in high end equipment called tractor tapes. Inch wide magnetic tapes that would be used for a while.

49:50 By in large, although Bell Labs has participated until fairly recently in data processing in quite a big way, AT&T never really quite trusted the Labs to do it right because here is where the money is. I can recall one occasion when during strike of temporary employees, a fill-in employee like from the Laboratories and so on, lost a day's billing tape in Chicago. And that was a million dollars. And that's generally speaking the money people did not until fairly recently trust Bell Labs to take good care of money, even though they trusted the Labs very well to make extremely reliable computing equipment for switches. The downtime on switches is still spectacular by any industry standards. The design for the first ones was two hours down in 40 years, and the design was met. Great emphasis on reliability and redundancy, testing.

51:35 Another branch of computing was for the government. The whole Whippany Laboratories [time check] Whippany, where we took on contracts for the government particularly in the computing era in anti-missile defense, missile defense, and underwater sound. Missile defense was a very impressive undertaking. It was about in the early '63 time frame when it was estimated the amount of computation to do a reasonable job of tracking incoming missiles would be 30 M floating point operations a second. In the day of the Cray that doesn't sound like a great lot, but it's more than your high end PCs can do. And the machines were supposed to be reliable. They designed the machines at Whippany, a twelve-processor multiprocessor, to no specs, enormously rugged, one watt transistors. This thing in real life performed remarkably well. There were sixty-five missile shots, tests across the Pacific Ocean ?...? and Lorinda Cherry here actually sat there waiting for them to come in. [laughter] And only a half dozen of them really failed. As a measure of the interest in reliability, one of them failed apparently due to processor error. Two people were assigned to look at the dumps, enormous amounts of telemetry and logging information were taken during these tests, which are truly expensive to run. Two people were assigned to look at the dumps. A year later they had not found the trouble. The team was beefed up. They finally decided that there was a race condition in one circuit. They then realized that this particular kind of race condition had not been tested for in all the simulations. They went back and simulated the entire hardware system to see if

its a remote possibility of any similar cases, found twelve of them, and changed the hardware. But to spend over a year looking for a bug is a sign of what reliability meant.

54:56 Since I'm coming up on the end of an hour, one could go on and on and on,

Crowd: go on, go on. [laughter]

55:10 Doug: I think I'd like to end up by mentioning a few of the programs that have been written at Bell Labs that I think are most surprising. Of course there are lots of grand programs that have been written.

I already mentioned the block diagram compiler.

Another really remarkable piece of work was eqn, the equation typesetting language, which has been imitated since, by Lorinda Cherry and Brian Kernighan. The notion of taking an auditory syntax, the way people talk about equations, but only talk, this was not borrowed from any written notation before, getting the auditory one down on paper, that was very successful and surprising.

Another of my favorites, and again Lorinda Cherry was in this one, with Bob Morris, was typo. This was a program for finding spelling errors. It didn't know the first thing about spelling. It would read a document, measure its statistics, and print out the words of the document in increasing order of what it thought the likelihood of that word having come from the same statistical source as the document. The words that did not come from the statistical source of the document were likely to be typos, and now I mean typos as distinct from spelling errors, where you actually hit the wrong key. Those tend to be off the wall, whereas phonetic spelling errors you'll never find. And this worked remarkably well. Typing errors would come right up to the top of the list. A really really neat program.

57:50 Another one of my favorites was by Brenda Baker called struct, which took Fortran programs and converted them into a structured programming language called Ratfor, which was Fortran with C syntax. This seemed like a possible undertaking, like something you do by the seat of the pants and you get something out. In fact, folks at Lockheed had done things like that before. But Brenda managed to find theorems that said there's really only one form, there's a canonical form into which you can structure a Fortran program, and she did this. It took your Fortran program, completely mashed it, put it out perhaps in almost certainly a different order than it was in Fortran connected by GOTOs, without any GOTOs, and the really remarkable thing was that authors of the program who clearly knew the way they wrote it in the first place, preferred it after it had been rearranged by Brendan. I was astonished at the outcome of that project.

59:19 Another first that happened around here was by Fred Grampp, who got interested in computer security. One day he decided he would make a program for sniffing the security arrangements on a computer, as a service: Fred would never do anything crooked. [laughter] This particular program did a remarkable job, and founded a whole minor industry within the company. A department was set up to take this idea and parlay it, and indeed ever since there has been some improvement in the way computer centers are managed, at least until we got Berkeley Unix.

60:24 And the last interesting program that I have time to mention is one by Ken Church. He was dealing with – text processing has always been a continuing ?. . . ? of the research, and in some sense it has an application to our business because we're handling speech, but he got into consulting with the department in North Carolina that has to translate manuals. There are millions of pages of manuals in the Bell System and its successors, and ever since we've gone global, these things had to get translated into many languages.

61:28 To help in this, he was making tools which would put up on the screen, graphed on the screen quickly a piece of text and its translation, because a translator, particularly a technical translator, wants to know, the last time we mentioned this word how was it translated. You don't want to be creative in translating technical text. You'd like to be able to go back into the archives and pull up examples of translated text. And the neat thing here is the idea for how do you align texts in two languages. You've got the original, you've got the translated one, how do you bring up on the screen, the two sentences that go together? And the following scam worked beautifully. This is on western languages. 62:33 Simply look for common four letter tetragrams, four letter combinations between the two and as best as you can, line them up as nearly linearly with the lengths of the two types as possible. And this very simple idea works like storm. Something for nothing. I like that.

63:10 The last thing is one slogan that sort of got started with Unix and is just rife within the industry now. Software tools. We were making software tools in Unix before we knew we were, just like the Moli‘ere character was amazed at discovering he'd been speaking prose all his life. [laughter] But then Kernighan and Plauger came along and christened what was going on, making simple generally useful and compositional programs to do one thing and do it well and to fit together. They called it software tools, made a book, wrote a book, and this notion now is abroad in the industry. And it really did begin all up in the little attic room where you [points?] sat for many years writing up here.

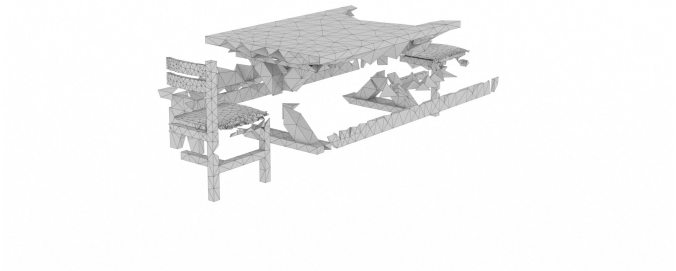
Oh I forgot to. I haven't used any slides. I've brought some, but I don't like looking at bullets and you wouldn't either, and I forgot to show you the one exhibit I brought, which I borrowed from Bob Kurshan. When Bell Labs was founded, it had of course some calculating machines, and it had one wonderful computer. This. That was bought in 1918. There's almost no other computing equipment from any time prior to ten years ago that still exists in Bell Labs. This is an integraph. It has two styluses. You trace a curve on a piece of paper with one stylus and the

other stylus draws the indefinite integral here. There was somebody in the math department who gave this service to the whole company, with about 24 hours turnaround time, calculating integrals. Our recent vice president Arno Penzias actually did, he calculated integrals differently, with a different background. He had a chemical balance, and he cut the curves out of the paper and weighed them. This was bought in 1918, so it's eighty years old. It used to be shiny metal, it's a little bit rusty now. But it still works.

66:30 Well, that's a once over lightly of a whole lot of things that have gone on at Bell Labs. It's just such a fun place that one I said I just could go on and on. If you're interested, there actually is a history written. This is only one of about six volumes, this is the one that has the mathematical computer sciences, the kind of things that I've mostly talked about here. A few people have copies of them. For some reason, the AT&T publishing house thinks that because they're history they're obsolete, and they stopped printing them. [laughter]

Thank you, and that's all.

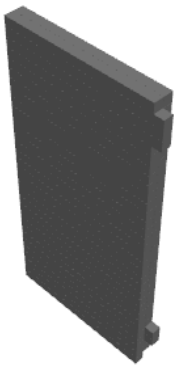
(Comments originally posted via Blogger.)



$$|\varphi(z)| \leq \pi^{\frac{1}{4}} e^{\frac{z\overline{z}}{2}} ||\varphi||_{SB}$$

- vaporland (April 9, 2008 7:04 PM) Fascinating post. The link to the original audio is broken, and google does not seem to be able to locate another copy.

Thanks very much for posting this - I really enjoyed it.



- David (April 10, 2008 9:38 PM) vaporland, try this link: doug97.rm
- Jack (April 21, 2008 5:53 AM) Where the transcript says, "and try permutations to see what you could do about reducing side ?. . . ?", what McIlroy said was "sidetone".
- Anonymous (December 19, 2010 12:11 PM) It's really a nice and helpful piece of information. I'm glad that you shared this helpful info with us. Please keep us informed like this. Thanks for sharing.
- Anonymous (February 9, 2011 5:17 PM) Nice and very informative post. Your point of view is more or less the same as main. Thanks!(Charlxtz)

$$SB[f](z) = \varphi(z) = \int_{\mathbb{R}} \mathcal{A}(z,u) f(u) du$$