Low-Cost IP Connectivity

Sun User Group, San Jose CA USA

December 9, 1991

Robert A. Sutterfield
Morning Star Technologies
bob@morningstar.com

ABSTRACT

We survey the history of packet-switched networking, with particular attention to the lowest cost access available at each stage of the growth and development of the Internet. We describe the development of SLIP and then PPP, and compare the two. PPP is observed to address many of the weaknesses of SLIP, and to be better suited to the task.

We describe a UNIX implementation of PPP. We compare its design with that of other and earlier efforts, and find this new approach to be superior in portability and flexibility. We also describe the "tunnel driver" interface to the UNIX IP layer, and show it to be the key to providing efficient on-demand intermittent IP connectivity. Some design weaknesses are discussed, but recognized as non-fatal flaws.

We discuss some early and obvious applications of low-cost connectivity, and present a few imaginative case studies. We then examine the current state of the low-cost IP infrastructure, including some vague handwaving concerning the future rate of growth and expansion of the IP connectivity vendors. Finally, we attempt to foresee the potential effects of low-cost connectivity on the computing and networking communities.

1 The Protocols Compared

1.1 A Brief History of the Internet

In the late 1960s, the United States Department of Defense Advanced Research Projects Agency began funding research into networking between dissimilar computers. The ARPAnet was a collection of custom-built packet switching nodes, connected by very expensive long-distance leased telephone lines. Hosts on the network were classical mainframes, sealed in glass rooms with their attendant drones. Hosts were connected, again by custom hardware and software, directly to the nearest packet switch.

A decade ago, with the adoption of IP, the wall fell down and the ARPAnet became a Catenet called the ARPA Internet. No longer were hosts the only thing that could be connected to the network; now networks could be connected as well. Once a network was connected, all the hosts on that network were connected as well. This was the beginning of the pervasive internetwork that has now found its way into every major technology company and educational institution in the USA, and many around the world.

Networking technology has expanded both upward and downward in cost and capability. At the high end, fat multigigabit data pipes provide well-funded researchers with a unique set of problems, mostly centered around the question, "what do we do with it all?" At the other end of the spectrum are the low-bandwidth "thin wires" that provide the same basic packet transport service, but at far lower cost. Using thin-wire protocols, it is possible to connect any computer as a full Internet peer, using only a voice-grade telephone line and a commodity modem.

1.2 IP On Thin Wires

The "thin wire" nomenclature is introduced in [Farber et al. 1984], where Farber, Delp, and Conte discussed strategies for connecting personal computers to the Internet. In one ambitious RFC, they approached everything from packet framing to TCP header compression to turning routers into application-level gateways acting as proxy agents for remote personal computers.

SLIP has its origins in the 3COM UNET TCP/IP implementation from the early 1980's. Around 1984, Rick Adams implemented SLIP for 4.2 Berkeley UNIX and Sun Microsystems workstations and released it on the world. It quickly became popular as a cheap and easy way to connect TCP/IP hosts and routers with serial lines.

It is a self-described "nonstandard", all of whose functions have been subsumed by PPP. PPP is the product of the Internet Engineering Task Force's Point to Point Working Group, and has been blessed by the IETF and Internet Activities Board as a standards-track protocol. SLIP was helpful for a time, and PPP's designers learned a lot of lessons from it, but its time has passed.

1.2.1 SLIP

Flag 0xC0	Data 0-MTU bytes	Flag 0xC0

Figure 1: full SLIP Packet Format

SLIP, described in RFC-1055 [Romkey 1988], is an extremely simple framing scheme for putting IP packets on a serial line: One special flag byte (0xC0) at the beginning, another at the end, as shown in figure 1. The initial flag is usually omitted, so that the ending flag signifies the boundary between two framefuls of data, as shown in figure 2.

Data 0-MTU bytes	Flag 0xC0

Figure 2: Minimal SLIP Packet Format

If one of the data bytes happened to be the same value as the flag byte, it is translated into a pair of characters (0xDB 0xDC) before transmission and then translated back upon reception, so that the data bytes aren't mistaken for the end of a packet. SLIP's main advantage is simplicity and therefore ease of implementation.

SLIP contains no provision for error detection, assuming that the upper level services will worry about that sort of stuff. It doesn't provide for link management tasks, nor for any protocol family besides IP. It contains no mechanism for negotiating values for any of the many things that can be different between hosts, such as network-level addresses or packet size or various sorts of protocol or data compression. It contains no link-level authentication or loop detection mechanism. These issues must be agreed upon in advance by the end-point systems, and often the decisions are embedded in code that isn't accessible to the user. For example, SLIP implementations that incorporate "VJ" TCP header compression [Jacobson 1990] typically must be told in advance not to do VJ when talking to SLIP implementations that can't understand what a header-compressed TCP/IP packet means.

A SLIP link is either open and successfully carrying IP packets, or it's closed. If a SLIP link is malfunctioning, then the reasons are potentially many and ill-defined. Did the administrators of each end fail to agree on frame size? VJ header compression? IP addresses? Considerable expertise and insight is helpful in debugging flaky connections. It helps to have a guru in the house.

1.2.2 PPP

Flag	Addr	Ctrl	Protocol	Data	FCS	Flag
0x7E	0xFF	0x03	0xXXXX	0-MTU bytes	16 or 32 bits	0x7E

Figure 3: Full PPP Packet Format

By contrast, PPP, described in RFC-1171 [Perkins 1990], is a much more robust, full-featured (some say bloated and over-featured) protocol. It adopts an established, well-understood framing scheme, HDLC, that includes error detection in every frame. Bad frames aren't even handed to the network layer, let alone to the session or application layers for their attention – they're simply dropped on the floor by the PPP implementation. Although the full PPP frame specification in figure 3 contains some per-frame overhead that's unnecessary for many links, PPP peers can agree upon various compression techniques, so the packets actually passed on a slow link will look more like those in figure 4.

Protocol	Data	FCS	Flag
0xXX	0-MTU bytes	16 bits	0x7E

Figure 4: Minimal PPP Packet Format

The RFC specifies ways that other protocol families like OSI, DECnet, Novell, XNS, etc. can share the link equally with IP, without requiring that they be encapsulated within IP packets. A PPP link can even be used as a protocol independent bridge. (Some of these specifications are still under development by interested parties in the IETF PPP Working Group.)

PPP specifies a link control protocol [Perkins et al. 1990], used for negotiating things like maximum frame size and HDLC-specific compression techniques. It also allows for control characters to be "escaped" (like the frame flag bytes described above) so that they can pass unmolested through hazardous datacomm equipment that might eat XON or XOFF characters or something else even more interesting. That control character escaping scheme is negotiable too, and can accommodate different special characters in each direction on the wire. The link control protocol provides for low-level authentication, link loop detection, and link quality monitoring that can be run in the midst of other traffic on the link.

For IP, the network control protocol [Perkins et al. 1990] allows the ends to negotiate their IP addresses, and whether they'll use TCP header compression. Dynamic address negotiation and assignment is a major management issue for hubs that provide service to many sporadically-attached remote hosts.

As a PPP link comes up, all the variables that may make it unsuccessful are subject to negotiation. A PPP implementation can easily report to its administrator "the link didn't come up because the peers couldn't agree upon their frame sizes," or whatever the cause. The most intimidating part of the PPP RFC, its elaborate finite state automaton, formalizes many of these link management tasks, distilling the wisdom of a link-level guru into an algorithmic process. This eases the human administrative burden of establishing and running the link, and empowers less-skilled users to solve the most common problems.

1.2.3 Performance Issues

There seems to be an impression abroad that PPP is slower than SLIP. Several times, someone has commented that they're considering both SLIP and PPP for a given application where either might be appropriate, but they've heard such a performance comparison story from "someone in the group." When asked in what way PPP might be slower than SLIP, they usually mention per-packet protocol overhead, but sometimes other issues.

Here are the usual responses to the most common objections. Remember that most people considering both SLIP and PPP are concerned about asynchronous dial-up IP between specific pairs of hosts that can easily run either SLIP or PPP, and don't care about connecting different manufacturers' routers with T1, or carrying any other protocol family besides IP.

0. "VJ" TCP header compression [Jacobson 1990] can be implemented over both (When SLIP is run with VJ, it's often called CSLIP), so that's not an issue for comparison. Just be sure yours does it.

0. Both PPP and SLIP have mechanisms for escaping characters in the data part of the frame that might be interpreted as part of the framing itself, so that's a wash too.

1. Unlike SLIP, PPP can operate even in the presence of in-band XON/XOFF ("software") flow control. Therefore, PPP may need to escape those characters in the data, thus requiring more octets to traverse the wire.

   This is an issue of the underlying transport, not of the protocol itself. In fact, this is an advantage in flexibility for PPP: the capability to escape almost any character is there if needed.

2. Since it has no option negotiation phase during link startup, SLIP can get under way more quickly, and therefore makes better use of precious and expensive telephone time. This is perhaps a valid critique – If both ends know *a priori* that they want to exchange certain protocol families and use certain addresses and certain types of compression, why not hot-wire the values and just do it?

   In our experience (running dial-up IP protocols between UNIX workstations), after the need to bring up the line occurs, the modems take 24 seconds to dial, connect a local call, train with V.32/V.42bis, and exchange the login/password handshake. After the login sequence is complete, the "other end"'s PPP process is started (so far, the requirements are the same whether with SLIP or PPP), and the PPPs typically negotiate their way up to readiness to pass IP packets within three seconds. Since the telephone company and the modems will absorb eight more time than the protocol, even on a local call, the extra time spent negotiating options is negligible.

Besides, PPP can be configured to insist upon any of the values and not negotiate them, or assign them on the fly. The ends can come more quickly to agreement.

3. SLIP has a much simpler framing scheme, when compared to PPP's kitchen sink approach, so its per-packet overhead is much lower, which means that the percentage of the link bandwidth devoted to paying cargo is much higher.

   If a PPP implements address/control field and protocol field compression, a typical PPP packet carries only three octets more baggage than a typical SLIP packet. Two of those three octets are used for FCS error calculations. It is is entirely an individual value judgement whether it matters that data is being delivered correctly at the link level, or whether that sort of worrying should be left to the higher-level protocols in the stack. The author has had a file corrupted inconveniently by non-UDP-checksummed NFS over SLIP, and he doesn't want it to happen again. He's willing to pay two octets per frame, even at 2400 baud.

   Also, the typical MTU in SLIP implementations is smaller (256 or or 576 or 1006, non-negotiable) than the typical MTU of a PPP link (1500, negotiable), so the protocol overhead octets can be amortized over a larger number of data octets. Of course, if a fast link to be used only for bulk data transfer, PPP can be instructed to negotiate a truly moby MTU and push the data:overhead ratio even higher. The user will realize actual benefits anywhere from negligible to negative, but it could be done.

4. Since SLIP's framing scheme is so simple, and since it specifies no finite state machine to implement and no error checking, it absorbs far less system resources than PPP.

   Cycles are cheap. Memory is cheap. A good PPP implementation can run in user space on a slightly aging UNIX workstation (a Sun SPARCstation-1) at its full async speed, doing all the frame assembly and disassembly and FCS and everything else, while using only a few hundred Kilobytes of memory and only a few percent of the host's CPU. Even running at T1 speeds, the daemon will only use a few percent of the CPU capacity (the tunnel driver will be busy, though). It can even be performing other gateway-oriented tasks like packet filtering and still provide good performance with negligible system overhead, but that's irrelevant to this point. If you're using an under-powered machine, you have other problems besides your communications overhead.

   As in argument 3 above, the peace of mind is worth the small cost in CPU and memory to calculate the FCS.

5. Since SLIP is so minimalist in its approach to the problem, it is much quicker to implement and verify than PPP, which seems unnecessarily complex and general for most people's problems.

   SLIP is indeed much easier to implement than PPP. But there now exist several freely available PPP implementations that may be used as the starting point for either free or commercial implementations. The engineering effort invested in PPP for IP now will pay off in the future when it will be much easier to plug in support for other protocol families. There's no reason to invest any more engineering effort in SLIP, except maybe for backwards compatibility.

   From the user's point of view, PPP provides the flexibility to more easily solve a wider range of configuration problems. And you may find out that you need Magic Numbers (loop detection), CHAP (authentication) or LQM (link quality monitoring) someday, even if you don't think so now.

6. But everyone does SLIP, and hardly anyone has PPP. It seems as if SLIP is the standard way of doing things.

   The SLIP RFC [Romkey 1988] itself describes SLIP as a non-standard, and lists several deficiencies. The PPP RFC [Perkins 1990] is the product of the Internet Engineering Task Force's Point to Point Protocol Working Group, and its introduction describes how it addresses each of those deficiencies. PPP is under active development and is experiencing directed evolution at the hands of a varied group of talented engineers; SLIP has stagnated for several years, having exhausted its potential. SLIP is John the Baptist, self-described as only pointing the way to far greater things to come. SLIP isn't fit to untie PPP's sandals.

7. If the links are already running SLIP, there is a significant effort required to change over to PPP. Both ends of each link must be able to run it. Why didn't they just design PPP to be upward compatible with SLIP, so we could phase it in gradually?

   SLIP is too simple to expand with the capabilities of PPP. By the time you've done that, you'd be basically running PPP anyway, so why not just start afresh?

1.2.4 Recommendations

New installations should start afresh with PPP. Systems designers should only use SLIP for interim backward compatibility with existing installations that are, for whatever reason, temporarily unable to run PPP. Soon, some clever people will create a PPP implementation for that other end.

2 UNIX Implementations of PPP

2.1 Earlier Efforts

Drew Perkins at Carnegie Mellon University wrote an implementation of asynchronous PPP for a VAX running 4.3BSD. This work was during the summer of 1989, while the initial PPP RFC was being worked out. In the spring of 1990, Brad Clements of Clarkson University ported that code to a Sun 386i running SunOS 4.0.1 and 4.0.2. That summer, Karl Fox of Morning Star Technologies ported that code to a Sun SPARCstation running SunOS 4.0.3c, then 4.1 and 4.1.1. Maintenance of that implementation has passed to Greg Christy of Quotron Systems, who has continued to update it as the PPP specification evolves. It is available via anonymous FTP from merit.edu. The same code may by now have been ported to various other UNIX variants as well, particularly Ultrix and NeXT's Mach.

Perkins's implementation of `if_ppp.c` was based upon the IP kernel interface (`if_sl.c`) developed by Rick Adams for his SLIP, later ported to 4.3BSD by Chris Torek. This approach put considerable knowledge about the PPP protocol inside the UNIX kernel, wrapped tightly around the interface to IP. For example, the same `pppopen()` in the `cdevsw[]` table contains code to manage the async control-character map, alongside code to manipulate the IP interface. This forces the protocol-handling code to deal directly with issues like interrupt priority levels and kernel memory management. This makes the code less portable between operating systems, the software more difficult for users to configure, and any bugs more difficult to squash.

Clements's modifications put VJ TCP header compression handling, FCS calculations, the asynchronous control-character map, HDLC address/control and PPP protocol field compression, and MRU/MTU handling inside STREAMS modules, with corresponding `ioctl()` hooks to manipulate them. Not all flavors of UNIX use STREAMS-based I/O, so this approach is also less portable than might be desired. Also, the aesthetic benefits of a STREAMS-based architecture are not universally accepted.

Also in early 1990, Torben Nielsen of the University of Hawaii wrote an implementation of synchronous PPP for Sun workstations, using the native serial ports with a custom HDLC driver. It required that the SunLink Internetwork Router product already have been installed, and just replaced the Sun "PTP" component. The PPP protocol and VJ TCP header compression handling were both in the `if_ppp()` module, the interface to the kernel's IP implementation.

Brixton Systems of Cambridge, MA has a product implementation of both asynchronous and synchronous PPP for Sun SPARCstations. Though this author knows few details of the Brixton design, scouts report that it is largely built of SunOS STREAMS modules. As before, a STREAMS implementation is not portable to all flavors of UNIX, so this approach is useful only on those systems that have *both* STREAMS I/O *and* BSD-style networking.

2.2 The Tunnel Driver

The "IP tunnel device driver" serves as the key to the new implementation's flexibility. It separates protocol handling functions from the kernel's knowledge of IP. Traditional networking interfaces, such as Ethernet or Token Ring, are implemented in hardware, with a device driver to carry the packets between the host's IP implementation and the interface itself. The hardware handles problems like backoff timers on a CSMA/CD medium, with which the kernel should not concern itself.

Rather than delivering packets to a hardware device, a tunnel driver makes them available on a file, such `/dev/tun0`. There, they may be read by a user-space daemon process, which has the responsibility for delivering the packets via the desired encapsulation scheme on whatever transport mechanism is available. One daemon might wrap the packets inside X.25 frames, as described in RFC-877 [Korb 1983], for transmission on a public data network. Another might wrap them inside PPP frames, as described in RFC-1171 [Perkins 1990], for transmission on a serial line. Another might prepare them for transmission across an SNA network, etc. The tunnel device driver is only responsible for presenting an interface to the UNIX kernel's IP.

Julian Onions of the Computer Science Department at Nottingham University wrote a tunnel driver in the fall of 1987, based on even earlier work by Phil Cockcroft of University College London, and used it for X.25 encapsulation. He saw it as useful for carrying IP over other networks besides X.25, but no results were published. Subsequent work by Toerless Eckert of the University of Erlangen refined the implementation, but it was still apparently used only to carry IP traffic across X.25 networks.

See Steven Bellovin's Usenix paper [Bellovin 1990] for a discussion of other issues. He proposes building virtual networks between and within the address spaces of other networks, and other uses for such a generic network interface.

2.3 A New Implementation

When compared with earlier efforts, the new daemon implementation is superior both in the functions it provides and the ease with which it may be ported to new platforms. The PPP protocol is entirely handled in a process in user space, and the tunnel driver provides a convenient and narrow interface to the kernel's IP layer. The daemon is not dependent upon new line disciplines or kernel architectures or STREAMS modules, which can hinder portability to different UNIX systems.

The daemon handles all aspects of the PPP protocol, and it manages the link state as well. It contains timers to support hanging up the telephone when the line goes idle, and it can dial the modem and reestablish connectivity when a packet needs to cross the link. Its state can be observed with `ps` because it manipulates its argument vector (`argv`). It can log its progress in the filesystem, by use of either `syslog` or its own mechanisms. It can respond sensibly to software signals, just like any other process. It can provide capabilities that would be difficult or dangerous to put into a kernel, or that would cause kernel processing to block inconveniently.

Protocol debugging, like every other task, is far quicker and less painful when implemented in a user-space process than in the kernel. Memory leaks and pointer overruns don't take down the entire machine. Variables and states are accessible, and the entire protocol engine can be run by single steps in a source-level debugger.

2.4 Weaknesses

This design admits of a few weaknesses, such as the number of times each packet must transit between the daemon and the kernel. However, modern workstations, particularly those with RISC architectures, provide inexpensive context switches, so this is a non-fatal flaw. The daemon can manage a PPP link at the full speed of the serial interface while using only a few percent of the CPU power of a SPARCstation-1. Interactive response time and latency are more constrained by the modem technology than by the implementation as a daemon and tunnel driver.

3 Applications and Effects

3.1 Observations

3.1.1 Application behavior

The scheme used to support on-demand dial-ups in Morning Star PPP depends upon network- and higher-layer protocol timeouts being typically longer than the time needed to allocate a modem, dial it, complete the call, login to the remote system, start up the PPP protocol handler there, and establish LCP and IPCP connectivity. This is a reasonable assumption, since a local call will typically require about 30 seconds between the daemon's receipt of the first 'bringup' packet and the establishment of IP-level connectivity (of that, more than 25 are consumed with dialing the call and training the modems). This scheme works even if the first remote modem is busy and the dialing *pppd* must try a second telephone number, so long as the dialer script is carefully written to recognize busy signals. The user's application will simply wait for the connection to be established and continue as if the connection had always been in place. This even works well when a link times out with active TCP sessions in place, such as a *telnet* connection to the remote end. When the user types, or when an application writes output, a new call is placed from either end and the session is re-established without IP, TCP, or the application above it noticing anything was amiss.

Unfortunately, if the connection cannot be re-established, TCP will time out and the application may fail ungracefully. This most often happens during times of resource contention (no free modems to place or answer the call) or acute connectivity problems (telephone system call-routing breakdown or extreme service degradation). The effect will be the same as if the machine at the 'other' end had crashed ungracefully, since the application will only be able to tell that it is unable to exchange packets with its partner.

3.1.2 Dealing with bandwidth limitations

The Internet's TCP/IP protocol suite is remarkably flexible and robust, with a wide variety applications for use over media ranging from multigigabit data superhighways to long-delay satellite channels to public data networks to local area networks to dial-up modems to amateur radio and other slow media. While by definition all IP-based applications may be transparently used over links traversing any combination of media, some applications are better suited than others to media with the relatively low bandwidth and moderate latency characteristics of asynchronous dial-up connections.

Applications with few problems on thin wires PPP over current-technology dial modems[1] provides satisfactory performance for most typical TCP/IP and UDP/IP wide-area applications. Especially with TCP header compression, users will experience useful file transfer throughput[2] and reasonable interactive response in applications such as *telnet*, *rlogin*, and the like. Plenty of bandwidth is available for network infrastructure applications such as SMTP electronic mail, NNTP network news, NTP clock synchronization, domain name service, etc.

Applications with some problems on thin wires The speed limitation of the dial-up medium becomes more apparent when larger amounts of data are transferred, particularly for interactive applications such as *xterm* or other display-oriented programs. While large bitmap editors and CAD packages will operate correctly, users may find that the modem simply isn't able to keep up with their demands for interactive responsiveness.

Applications requiring special care on thin wires Applications that are extreme users of communications bandwidth, such as Sun's Network File System (NFS), should be used with care over dial-up links. Not only is NFS able to easily overwhelm a modem link, it is a UDP-based application, so the user will not enjoy the packet overhead reduction benefits of TCP header compression. While performance may be marginally acceptable for read-only file access, NFS's synchronous write semantics will make file creation and writing noticeably slower than on a local file system. Attempting to execute a program stored on a remotely-mounted filesystem will also have this problem, since paging will occur from the remote file.

Fortunately, there are usually other approaches, more appropriate for wide-area networking over low-bandwidth links, to solving the problems that NFS addresses. Use Berkeley's *rdist* system to keep the important segments of two sites' file structures in reasonable synchrony. Also, use a package like *ange-ftp.el*[3] in your GNU Emacs to automatically manage FTP connections that are transparently associated with your editing buffers. This gives the appearance of the ability to edit files while they reside on a remote machine, without needing to use NFS to mount the remote filesystems locally, and without needing a constantly-open physical connection.

If, even after the above attempts at dissuasion, you still think that you absolutely must use NFS, increase the mount request's *timeo* timeout parameter to accommodate network delays.

3.2 Typical Configurations

3.2.1 Two Workstations

Two distant workstations might arrange a private dial-up PPP link, much as they would use UUCP to exchange files, mail, and news. The PPP link would use the same modems as UUCP, and the effort involved in setting it up is about the same as that for UUCP. They would benefit by interactive access, simultaneous bidirectional activity, and the potential for using the full range of TCP/IP applications. They would even gain in bit efficiency, since although TCP/IP isn't quite as concerned with line bandwidth frugality as Zmodem or Kermit, it's far better than UUCP's "g"

[1] CCITT V.32 (9600) or V.32bis (14400) carrier, with V.42 error correction and V.42bis data compression, and the host serial interface latched at 38400

[2] V.32/V.42bis yields 1.7-2.8 Kbytes/sec, and V.32bis/V.42bis yields 2.4-3.5 Kbytes/sec for a typical large binary data file. Over Telebit T1600 V.32/V.42bis modems latched at 38400, the daemon can receive 25224 bits/sec and simultaneously send 26510 bits/sec

[3] Written by Andy Norman, ange@hplb.hpl.hp.com, available via anonymous FTP from ftp.gnu.ai.mit.edu:/ange-ftp/ange-ftp.el.Z

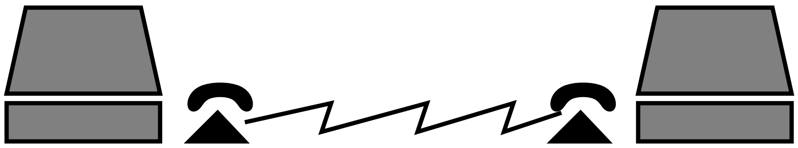

Figure 5: Two Workstations

protocol.

Some minicomputer and mainframe manufacturers provide a modem or even a small computer for front-end diagnostic access to their systems in the field. If that remote front-end is a board-level UNIX workstation, it could be linked via PPP into the diagnostic tools on the manufacturer's home office network for quicker problem resolution.

3.2.2 Workstation to LAN

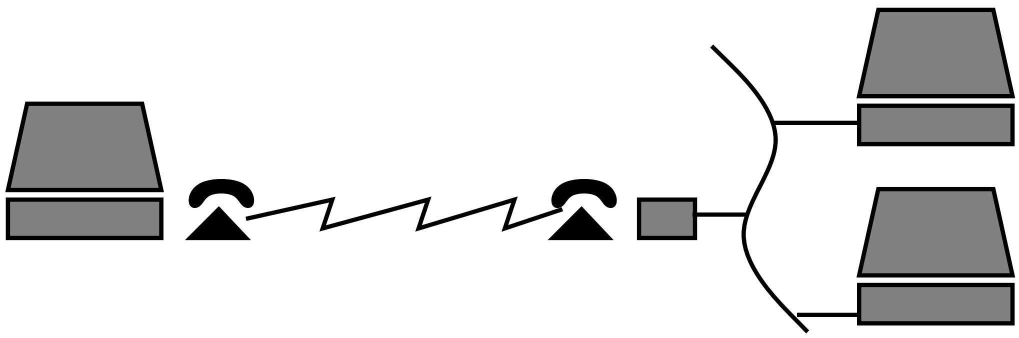

Figure 6: Workstation to LAN

Many forward-thinking companies using workstations in the office see benefits in providing their engineers with similar workstations for use at their homes, particularly since the home machines can now be easily connected with the office network. It's much less expensive to install a voice telephone line into a home than a dedicated leased data line, and there's less of a wait for installation. ISDN will change all the price/performance tradeoffs.

Students who purchase low-cost workstations upon matriculation should be able to connect them with the campus network, thence to the rest of the Internet. This is becoming common in university-provided housing where it is practical to extend fiber or twisted pair to each dormitory room, but off-campus residents can now receive the same services.

With the advent of "laptop" or at least "portable" workstations, migratory professionals like field support and marketing personnel can have full-function access to the home office databases, perhaps even over cellular telephone calls. As commercial IP network points of presence (POPs) become more ubiquitous, the traveling workstation user may not even need to dial directly into the home office network, but will use local access points with cheaper telephone charges.

3.2.3 LAN to LAN

Widely separated offices of the same organization can use dial-up IP to form an enterprise-wide network at much lower cost than if dedicated leased lines and routers were required.

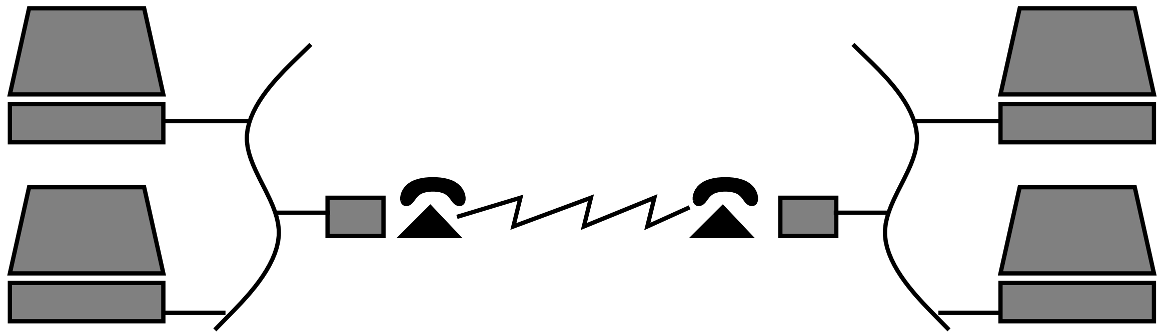

Figure 7: LAN to LAN

As companies enter into cooperative product development agreements, they can link their respective LANs for technology transfer.

While moving equipment to new facilities, dial-up IP can provide a quick, cheap, flexible lifeline of connectivity before permanent networking facilities are fully installed.

3.2.4 LAN to Internet

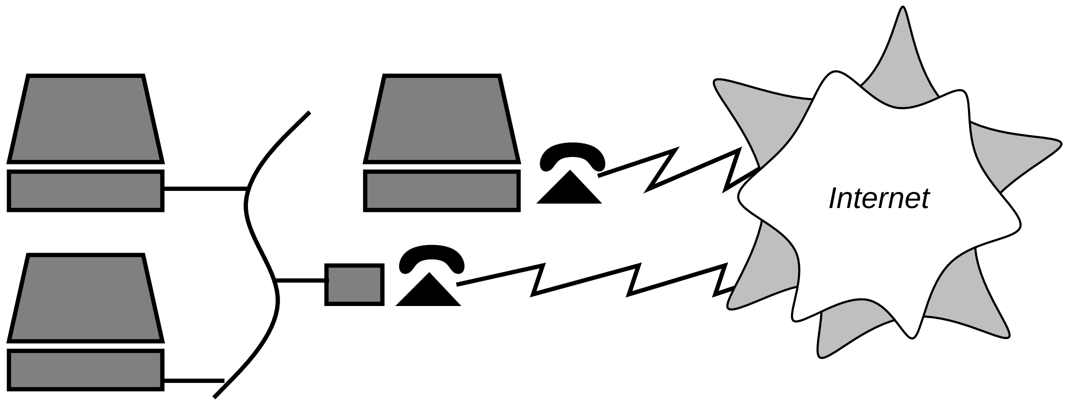

Figure 8: LAN to Internet

Small colleges and universities can "get their feet wet" in networking by connecting to their regional NSFnet tributary via an inexpensive dial-up link, planning to commit the resources later to install a dedicated leased line and router. The low cost of entry should be an effective marketing tool for the commercial IP connectivity vendors. Companies can use a workstation with a modem to provide an initial Internet gateway for their entire network, through which they have full access to all the Internet's facilities. Once users become accustomed to the services and the low-speed link becomes saturated, the more expensive alternatives will become more easily justifiable. A small company will outgrow a dial-up link in about a year, but during that time they'll see the benefits and their employees will become dependent upon it, and they'll never be able to unplug again.

3.3 Case Studies

3.3.1 Anonymous-Access Software Archives

A major university's computer science department has for several years maintained free software archives for access via anonymous FTP across the Internet, and for anonymous dial-up UUCP for anyone with a UNIX machine and a modem. This is a valuable community service, and has inspired other sites to begin providing similar facilities.

They have lately begun offering their services over dial-up PPP links. As with the anonymous UUCP logins, their modem telephone number and login sequence are widely published. Anyone running PPP can establish an IP connection and browse their archives via anonymous FTP, just as anyone with a permanent Internet connection. Security concerns are minimal since no routes are propagated, so the remote access is limited to the machine answering the call and hosting the anonymous PPP login. The archive server's PPP daemon filters out all packet types except ICMP and those involved in the FTP transaction itself, to avoid cracking attempts via *telnet* or *rlogin* or denial-of-service attacks via *routed*.

They're considering allowing remote NFS mounts too, just as they do across the Internet.

3.3.2 Medical Information

A company sells a medical information system that uses workstations for data collection at the patient bedside and database servers in the hospital MIS center. When a patient has a problem and an intern calls a doctor at home at 3:30am, the doctor needs access to the most current data about that patient. Using PPP, the company is extending their information services, formerly available only on the hospital LAN, into the doctor's home. Each doctor has a low-cost color workstation running Open Windows to display the same picture the intern sees at the bedside.

The database query engine and graphic generator currently run on large Sun systems at the hospital, and the PPP connection carries PostScript images to the doctor's home. Future revisions of the software may support remote queries and generate the images on the workstation, so that the PPP connection would carry the database transactions rather than the resultant graphics, but this hasn't yet been shown to yield improved performance to the physician, nor improved care to the patient.

3.3.3 Résumé Processing

A company provides database software for automated résumé handling. Clerks optically scan incoming paper résumés for entry into a database, with keywords, educational and work history, etc. When a position becomes open, the requisition is pattern-matched against the database to screen for qualified applicants, which are then forwarded to personnel officers for consideration.

Some users of this system are geographically diverse, with multiple employment and hiring locations. The widespread hiring-database servers are linked by dial-up modems. The system was initially implemented with a custom serial protocol designed specifically for queries by this particular database. When the query protocol was converted to run over TCP streams over IP networks, the dial-up links were converted to PPP and became useful for customer support, remote printing, electronic mail, and everything else alongside the database queries.

Rather than being restricted to one application, the modem links carrying IP traffic have become part of the corporate infrastructure. Dial-Up PPP allowed the system's developers to separate the networking issues from their product, and yields increased flexibility for the customers.

3.3.4 Air Traffic Control

A group at NASA is developing an Air Traffic Control automation system for the FAA, consisting of a truly distributed set of tools running on high speed color graphic UNIX workstations, and communicating via TCP sockets. Using on-demand dial-up IP enabled developers spread across the continent to collaborate and develop the application closer to schedule and more nearly within budget. Many of those developers are contractors without Internet access, so dial-up IP was the only practical solution for providing connectivity to their offices.

Dial-up PPP was deployed in three gradual phases:

Phase 1 PPP simply replaced UUCP for electronic mail and file transfer. UUCP could even continue to run over a TCP connection. Clever shell scripts that had already been worked out to keep remote source code areas could continue to work unmodified, until the users developed confidence in the new capabilities and the shell scripts could be replaced with applications like *rdist*.

Phase 2 The various parts of the distributed application were linked together so that widely-separated engineers could participate in air traffic simulations and cooperative debugging sessions. This also provided a realistic approximation of the operational environment the system will encounter when deployed.

Phase 3 Certain ATC monitoring services, currently provided only over a dedicated leased line from one location to another, will soon be made available to any authorized machine via dial-up PPP.

3.4 The Dial-Up IP Infrastructure

3.4.1 What Exists Today

In ages past, dial-up IP was used between well-prepared individuals in narrow applications. Hackers with machines at home were highly motivated to install SLIP on their office machines, and they were tolerant of the difficulties in using and maintaining the connection. They put up with slow, unreliable connections because of the benefits to be gained by connectivity.

In the months since Telebit's NetBlazer was introduced, several network providers have entered the market to sell dial-up IP connectivity. For a fee, they guarantee a level of service, availability, performance, and support. Their customers tolerate the low bandwidth of a dial-up connection because their monthly expenses are far below the prevailing rates of only a few months before, which required a leased line and a dedicated router. Competitors in the IP marketplace include Performance Systems International's PSInet, Uunet Communications's AlterNet, CERFnet, BARRnet, MSEN, and more startups every month. Some of the NSFnet regional and mid-level tributaries have begun offering dialup service. The European networking community is adopting dial-up IP as their base level of connectivity support, and several countries' network providers are cooperating to provide uniform quality service.

3.4.2 Prognostications

Though other technologies like ISDN are coming someday, they will never be as ubiquitous as simple voice-grade telephone lines. With implementations of the TCP/IP protocol suite becoming available for lower and lower cost computers, with both free and commercial dial-up implementations becoming more easily accessible to less technical users, and with more volume to be sold and profit to be made in the business of providing connectivity, we can only expect dial-up services to blossom.

3.5 Effects On Computing, And On The World Community

Low-cost dial-up IP connectivity is doing for networking what personal computers did for general purpose computing. Just as computational power moved out of the glass-walled machine room and onto the individual's desktop, dial-up IP is carrying the power of the network away from the big research centers, into the small schools and businesses, and into people's homes. While well-funded researchers are extending the peak of the networking pyramid ever higher into the stratospheric Gigabit realms, dial-up IP is widening the base to include a far larger number of users. Until now, there have been ill feelings and envy in the unconnected computing community, and almost a tangible snobbish attitude among the elite few who had already "arrived" on the Internet. Those divisions are disappearing.

The basic internetworking infrastructure is now available to a much larger clientele, not only in the computing and technical disciplines, but for other purposes as well. Since connectivity is available at a far lower cost, it is newly accessible to organizations and individuals with more urgent priorities for their funding. Small colleges, public schools, non-profit organizations, and others in the West benefit from sharing contacts and program content. In most of the world, the telecommunications infrastructure has not developed to the point of providing fiber-optic cable to every doorstep. Gigabits aren't needed where rice is lacking! Humanitarian relief organizations use the networks to communicate between front-line workers and their resource-rich supporters in the West. Though the world's problems aren't the sort that can be directly solved by low-cost networking, or any sort of technological "quick fix", technology is making those relief efforts more efficient and effective. Data networking is taking its place among the list of "enabling technologies" put to work for those in need.

References

[Bellovin 1990] S.M. Bellovin, *Pseudo-Network Drivers and Virtual Networks*, Proc. Winter USENIX Conference, Washington DC, 229-244, 1990.

[Korb 1983] J. T. Korb, RFC-877 *A Standard for the Transmission of IP Datagrams Over Public Data Networks*, Purdue, September 1983.

[Farber et al. 1984] D. J. Farber, G. S. Delp, and T. M. Conte, RFC-914 *A Thinwire Protocol for connecting personal computers to the INTERNET*, University of Delaware, September 1984.

[Romkey 1988] J. Romkey, RFC-1055 *A Nonstandard For Transmission Of IP Datagrams Over Serial Lines: SLIP*, BBN, June 1988.

[Jacobson 1988] V. Jacobson and R. Braden, RFC-1072 *TCP Extensions for Long-Delay Paths*, LBL and ISI, October 1988.

[Jacobson 1990] Van Jacobson, RFC-1144 *Compressing TCP/IP Headers for Low-Speed Serial Links*, Lawrence Berkeley Laboratory, February 1990.

[Waitzman 1990] David Waitzman, RFC-1149 *A Standard for the Transmission of IP Datagrams on Avian Carriers*, BBN, April 1990.

[Perkins 1990] Drew D. Perkins, RFC-1171 *The Point-to-Point Protocol for the Transmission of Multi-Protocol Datagrams Over Point-to-Point Links*, CMU, July 1990.

[Perkins et al. 1990] Drew D. Perkins and Russ Hobby, RFC-1172 *The Point-to-Point Protocol (PPP) Initial Configuration Options*, CMU and UC Davis, July 1990.