A Brief History of NTP Time: Memoirs of an Internet Timekeeper[1,2]

David L. Mills, *Fellow ACM, Fellow IEEE*[3]

Abstract

This paper traces the origins and evolution of the Network Time Protocol (NTP) over two decades of continuous operation. The technology has been continuously improved from hundreds of milliseconds in the rowdy Internet of the early 1980s to tens of nanoseconds in the Internet of the new century. It includes a blend of history lessons, technology milestones and series of experiments that shape, define and record the early history of the Internet and NTP.

This narrative is decidedly personal, since the job description for an Internet timekeeper is highly individualized and invites very few applicants. There is no attempt here to present a comprehensive tutorial, only a almanac of personal observations, eclectic minutiae and fireside chat. Many souls have contributed to the technology, some of which are individually acknowledged in this paper, the rest too numerous left to write their own memoirs.

Keywords: computer network, time synchronization, technical history, algorithmic memoirs

1. Introduction

An argument can be made that the Network Time Protocol (NTP) is the longest running, continuously operating, distributed application in the Internet. As NTP is in its third decade, it is of historic interest to document the origins and evolution of the architecture, protocol and algorithms. Not incidentally, NTP was an active participant in the early development of Internet technology and its timestamps recorded many milestones in measurement and prototyping programs.

This paper documents significant milestones in the evolution of computer network timekeeping technology over four generations of NTP to the present. The NTP software distributions for Unix, Windows and VMS have been maintained by a corps of over four dozen volunteers at various times. There are too many to list here, but the major contributors are revealed in the discussion to follow. The current NTP software distribution, documentation and related materials, newsgroups and links are on the web at www.ntp.org. In addition, all papers and reports by this author and cited herein are in PostScript and PDF at www.eecis.udel.edu/~mills. Further information, including executive summaries, project reports and briefing slide presentations are at www.eecis.udel.edu/~mills/ntp.html.

There are three main threads interwoven in the following. First is a history lesson on significant milestones for the NTP specifications, implementations and coming-out parties. These milestones calibrate and are calibrated by developments elsewhere in the Internet community. Second is a chronology of the algorithmic refinements leading to better and better accuracy, stability and security that continue to the present. These algorithms represent the technical contributions as documented in the references. Third is a discussion of the various proof-of-performance demonstrations and surveys conducted over the years, each attempting to outdo the previous in calibrating the performance of NTP in the Internet of the epoch. Each of these three threads winds through the remainder of this narrative.

2. On the Antiquity of NTP

NTP's roots can be traced back to a demonstration at NCC 79, believed to be the first public coming-out party of the Internet sending data, speech and facsimile messages over a transatlantic satellite network. However, it was not until 1981 that the synchronization technology was documented in the now historic Internet Engineering Note series as IEN-173 [53]. The first specification

1. Sponsored by: DARPA Information Technology Office Order G409/J175, Contract F30602-98-1-0225, and Digital Equipment Corporation Research Agreement 1417.

2. This document has been submitted for publication. It is provided as a convenience to the technical community and should not be cited or redistributed.

3. Author's address: Electrical and Computer Engineering Department, University of Delaware, Newark, DE 19716, mills@udel.edu, http://www.eecis.udel.edu/~mills.

of a public protocol developed from it appeared in RFC-778 [52]. The first deployment of the technology in a local network was as an integral function of the Hello routing protocol documented in RFC-891 [50], which survived for many years in a network prototyping and testbed operating system called the Fuzzball [44].

NTP was not then and is not now the only synchronization scheme available in the Internet. Other mechanisms have been specified in the Internet protocol suite to record and transmit the time at which an event takes place, including the Daytime [58] and Time [60] protocols and ICMP Timestamp [66] message defined as Internet Standards, as well as the IP Timestamp option [66].

The Unix *timed*, which arrived in 1984 [7], was designed for use on a local Ethernet. It uses an election algorithm to designate a single host as master and the remaining hosts synchronized to it as slaves. The Digital Time Synchronization Service (DTSS) [6], which was adopted by the Enterprise community, uses a hierarchy of time providers, servers and clerks similar to the NTP stratum model. A scheme with features similar to NTP is described in [57]. It is intended for multi-server LANs where each of a set of possibly many time servers determines its time offset relative to each of the other servers in the set using periodic timestamped messages, then determines the local clock correction using a fault-tolerant averaging algorithm. However, none of these schemes have crafted data grooming and clock discipline algorithms as later developed for NTP.

What later became known as NTP Version 0 was implemented in 1985, both in Fuzzball by this author and in Unix by Louis Mamakos and Michael Petry at University of Maryland. Fragments of the Unix code survive in the software running today. RFC-958 contains the first formal specification of this version [47], but it did little more than document the NTP packet header and offset/delay calculations still used today. Considering the modest speeds of networks and computers of the era, the nominal accuracy that could be achieved on an Ethernet was in the low tens of milliseconds. Even on paths spanning the Atlantic, where the jitter could spike one second, the accuracy was generally better than 100 ms.

Version 1 of the NTP specification was documented three years later in RFC-1059 [45]. It contained the first comprehensive specification of the protocol and algorithms, including primitive versions of the clock filter, selection and clock discipline algorithms. The design of these algorithms was guided largely by a series of experiments, documented in RFC-956 [49], in which the basic theory of the clock filter algorithm was developed and refined. This was the first version which defined the use of client/server and symmetric modes and, of course, the first version to make use of the version field in the header.

A transactions paper on NTP Version 1 appeared in 1991 [40]. This was the first paper that exposed the NTP model, including the architecture, protocol and algorithms, to the general engineering community. While this model is generally applicable today, there have been a continuing series of enhancements and new features introduced over the years, some of which are described in following sections.

The NTP Version 2 specification followed as RFC-1119 in 1989 [43]. A completely new implementation slavish to the specification was built by Dennis Fergusson at University of Toronto. This was the first RFC in PostScript and as such the single most historically orthogonal document in the RFC publishing process. It was the first to include a formal model and state machine describing the protocol and pseudo-code defining the operations. It introduced the NTP Control Message Protocol for use in managing NTP servers and clients, and the cryptographic authentication scheme based on symmetric-key cryptography, both of which survive to the present day.

There was considerable discussion during 1989 about the newly announced DTSS. DTSS and NTP communities had much the same goals, but somewhat different strategies for achieving them. One problem with DTSS, as viewed by the NTP community, was a possibly serious loss of accuracy, since the DTSS design did not discipline the clock frequency[4]. The problem with the NTP design, as viewed by the DTSS community, was the lack of formal correctness principles in the design process. A key component in the DTSS design upon which the correctness principles were based was an agreement algorithm invented by Keith Marzullo in his dissertation and described in [18].

In the finest Internet tradition of stealing good ideas, the Marzullo algorithm was integrated with the existing suite of NTP data grooming algorithms, including the filter, clustering and combining algorithms, which the DTSS design lacked. However, the Marzullo algorithm

4. NTP is not the only scheme to discipline the frequency; this is done also in the schemes described by Liao [13] and Paxton [57], although the latter does not operate in real time.

in its original form produced excessive jitter and seriously degraded timekeeping quality over typical Internet paths. The algorithm, now called the intersection algorithm, was modified to avoid this problem. The resulting suite of algorithms has survived substantially intact to the present day, although many modifications and improvements have been made over the years.

In 1992 the NTP Version 3 specification appeared as RFC-1305 [37], again in PostScript and now running some 113 pages. As the IETF insisted on ASCII, the official IETF document is the Corel Ventura document source with brutalized equations. The specification included an appendix describing a formal error analysis and an intricate error budget including all error contributions between the primary reference source over intervening servers to the eventual client. This provided the basis to support maximum error and estimated error statistics, which provide a reliable characterization of timekeeping quality, as well as a reliable metric for selecting the best from among a population of available servers. As in the Version 2 specification, the model was described using a formal state machine and pseudocode. This version also introduced broadcast mode and included reference clock drivers in the state machine.

Lars Mathiesen at University of Copenhagen carefully revised the version 2 implementation to comply with the version 3 specification. There was considerable give and take between the specification and implementation and some changes were made in each to reach consensus, so that the implementation was aligned precisely with the specification. This was a major effort which lasted over a year during which the specification and implementation converged to a consistent formal model.

In the years since the version 3 specification, NTP has evolved in various ways adding new features and algorithm revisions while still preserving interoperability with older versions. Somewhere along the line, it became clear that a new version number was needed, since the state machine and pseudo-code had evolved somewhat from the version 3 specification, so it became NTP Version 4. The evolution process was begun with a number of white papers, including [33] and [30].

Subsequently, a simplified version 4 protocol model was developed for the Simple Network Protocol (SNTP) version 4 in RFC-2030 [28]. SNTP is compatible with NTP and specified for the IPv4, IPv6 and OSI protocol stacks[5]. However, SNTP does not include the crafted mitigation and discipline algorithms. These algorithms are not necessary in an implementation intended solely for a PC client or a server synchronized to an external time source, such as a GPS receiver. SNTP version 4 has appeared in Windows XP and is used in several standalone NTP servers integrated with GPS receivers.

There is a certain sense of the radio amateur in the deployment of NTP around the globe. Certainly, each new country found running NTP was a new notch in the belt. A particularly satisfying conquest was when the national standards laboratory of a new country came up an NTP primary server connected directly to the national time and frequency ensemble. Internet timekeepers Judah Levine at NIST and Richard Schmidt at USNO deployed public NTP primary servers at several locations in the US and overseas. There was a period where NTP was well lit in the US and Europe but dark elsewhere in South America, Africa and the Pacific Rim. Today, the Sun never sets or even gets close to the horizon on NTP. The most rapidly growing populations are in Eastern Europe and South America, but the real prize is a new one found in Antarctica. Experience in global timekeeping is documented in [27].

One of the real problems in fielding a large, complex software distribution is porting to idiosyncratic hardware and operating systems. There are now over two dozen ports of the distribution for just about every hardware platform running Unix, Windows and VMS marketed over the last twenty years, some of them truly historic in their own terms. Various distributions have run on everything from embedded controllers to supercomputers. Maintaining the configuration scripts and patch library is a truly thankless job and getting good at it may not be a career enhancer. Volunteer Harlan Stenn currently manages this process using modern autoconfigure tools. New versions are tested first in our research net DCnet, then in bigger sandboxes like CAIRN and 6BONE and finally put up for public release at www.ntp.org. The bug stream arrives at bugs@ntp.org.

At this point the history lesson is substantially complete. However, along the way several specific advancements need to be identified. The remaining sections of this paper discuss a number of them in detail.

3. Autonomous Authentication

Some time around 1985 Project Athena at MIT was developing the Kerberos security model, which provides cryptographic authentication of users and services. Fundamental to the Kerberos design is the ticket used to

5. An implementation of NTP using the OSI protocol stack over X.25 is described by Crowcroft [5].

access computer and network services. Tickets have a designated lifetime and must be securely revoked when their lifetime expires. Thus, all Kerberos facilities had to have secure time synchronization services. While the NTP protocol contains specific provisions to deflect bogus packets and replays, these provisions are inadequate to deflect more sophisticated attacks such as masquerade. In order to deflect such attacks, NTP packets can be authenticated using symmetric key cryptography with keyed message digests and private keys. The original scheme used the Digital Encryption Standard operating in Cipher Block Chaining mode (DES-CBC).

Provision of DES-based source authentication created problems for the public software distribution. Due to the International Trade in Arms Regulations (ITAR) at the time, DES could not be included in NTP distributions exported outside the US and Canada. Initially, the way to deal with this was to provide two versions of DES in the source code, one operating as an empty stub and the other with the algorithm but encrypted with DES and a secret key. The idea was that, if a potential user could provide proof of residence, the key was revealed. Later, this awkward and cumbersome method was replaced simply by maintaining two distributions, one intended for domestic use and the other for export. Recipients were placed on their honor to fetch the politically correct version.

However, there was still the need to authenticate NTP packets in the export version. Louis Mamakos at University of Maryland adapted the MD5 message digest algorithm for NTP. This algorithm is specifically designed for the same function as the DES-CBC algorithm, but is free of ITAR restrictions. In NTP Version 4 the export distribution has been discontinued and the DES source code deleted; however, the message digest algorithm interface is compatible with the OpenSSL cryptographic library widely used in the Internet of today. Presumably, OpenSSL Blowfish-CBC or IDEA-CBC could be used according to fancy.

While message digest source authentication has worked well, it requires secret keys, which complicates key distribution and, especially for multicast-based modes, is vulnerable to compromise. Public-key cryptography simplifies key distribution, but can severely degrade timekeeping quality.

The Internet Engineering Task Force (IETF) has defined several cryptographic algorithms and protocols, but these require persistent state, which is not possible in some NTP modes. Some appreciation of the problems is apparent from the observation that secure timekeeping requires secure cryptographic media, but secure media require reliable lifetime enforcement [23]. The implied circularity applies to any secure time synchronization service, including NTP [22].

These problems were addressed in NTP Version 4 with a new security model and protocol called Autokey [19]. Autokey uses a combination of public-key cryptography and a pseudo-random keystream. Since public-key cryptography hungers for large chunks of processor resources and can degrade timekeeping quality, the algorithms are used sparingly to sign and verify time values, while the much less expensive keystream is used to authenticate packets relative to the signed values. Furthermore, Autokey is completely self-configuring, so that servers and clients can be deployed and redeployed in an arbitrary topology and automatically exchange signed values without manual intervention.

Autokey uses industry standard certificates and trusted authorities; however, certificate trails are a middleman hazard in an ad-hoc network, which NTP surely is. To avoid a middleman masquerade, a number of cryptographic challenge-response identity schemes have been incorporated in the design. In general, the goals of the schemes are that clients cannot masquerade as servers and servers cannot masquerade as trusted authorities (TA), but they differ somewhat on how to achieve these goals. To the extent that identity can be verified without revealing the group key, the schemes are properly described as zero-knowledge proofs. Two of these schemes are described below.

The IFF scheme [63] is intended for servers operated by national laboratories. The servers use a private group key and provide a password encrypted client key on request. The servers share the same group key, but it is not neccesary that they protect each other from masquerade. The clients do not know the group key, so cannot masquerade as legitimate servers. The University of Delaware primary NTP time servers use this scheme along with an automated mail system to retrieve encrypted client keys.

The MV scheme [56] is intended for the most challenging scenarios where it is neccesary to protect against both TA and server masquerade. The private values used by the TA to generate the cryptosystem are not available to the servers and the private values used by the servers to generate client keys are not available to the clients. However, a client can verify a server has the correct group key even though neither the client nor server know the group key, nor can either manufacture a client key acceptable to any other client. A further feature of this scheme is that the TA can collaborate with the serv-

ers to revoke a client key without changing other client keys.

Further information is available at www.eecid.udel.edu/~mills/identity.html.

4. Autonomous Configuration

It became clear as the NTP development continued that a most valuable enhancement would be the capability for a number of clients and servers to automatically configure and deploy in an NTP subnet delivering the best timekeeping quality, while conserving processor and network resources. Not only would this avoid the tedious chore of engineering specific configuration files for each server and client, but it would provide a robust response and reconfiguration scheme should components of the subnet fail. The DTSS model described in [6] goes a long way to achieve this goal, but has serious deficiencies, notably the lack of cryptographic authentication.

In NTP Version 3, configuration files had to be constructed manually using information found in the lists of public servers at www.ntp.org, although some sites partially automated the process using crafted DNS records. Where very large numbers of clients are involved, such as in large corporations with hundreds and thousands of personal computers and workstations, the method of choice is broadcast mode, which was added in NTP Version 3, or IPv4 multicast mode and IPv6 broadcast mode, which were added in NTP Version 4.

However, in NTP Version 3 clients did not send to servers, so there was no way to calibrate and correct for the server-client propagation delay, nor was there a way to initialize the Autokey protocol and run the identity schemes. This is provided in NTP Version 4 by a protocol modification in which the client, once receiving the first broadcast packet, executes a volley of client/server exchanges in order to calibrate the delay and run the Autokey protocok, then reverts to listen-only mode.

In spite of the protocol modification, broadcast mode provides somewhat less accuracy than client/server mode, since it does not track variations due to routing changes or network loads. In addition, it is not easily adapted for autonomous deployment. In NTP Version 4 a new Manycast mode was added where a client sends to an IP multicast group address and any server listening on this address responds with a unicast packet, which then mobilizes an association in the client. The client processes the the unicast packets from a few to a dozen servers, then winnows the population down to three using the NTP mitigation algorithms.

Manycast mode has the potential to allow at least moderate numbers of servers and clients to nucleate about a number of primary servers, but the full potential for autonomous deployment can be realized only using symmetric mode, where the NTP subnet can grow and flex in fully distributed and dynamic ways. In his dissertation Ajit Thyagarajan examines a class of heuristic algorithms that may be useful management candidates. Meanwhile, the quest for new technology continues.

5. Radios, we have Radios

For as many years as NTP has ticked on this planet, the definitive source for public NTP servers in the Internet has been a set of public lists, one for primary servers and the other for secondary servers, maintained at www.ntp.org. Each server in those tables is operated as a public service and maintained by a volunteer staff. Primary (stratum 1) servers have up to several hundred clients and a few operated by NIST and USNO have many thousands. A primary server requires a primary reference source, usually a radio or satellite receiver or modem. Following is a history lesson on the development and deployment of NTP primary servers in the Internet.

The first use of radios as a primary reference source was in 1981 when a Spectracom WWVB receiver was connected to a Fuzzball at COMSAT Laboratories in Clarksburg, MD [52]. This machine provided time synchronization for Fuzzball LANs in the US, UK, Norway, Germany and Italy. These LANs were used in the DARPA Atlantic Satellite program for satellite measurements and protocol development. Later, the LANs were used to watch the national power grids of the US, UK and Norway swish and sway over the heating and cooling seasons [48].

DARPA purchased four of the Spectracom WWVB receivers, which were hooked up to Fuzzballs at MIT Lincoln Laboratories, COMSAT Laboratories, USC Information Sciences Institute, and SRI International. The radios were redeployed in 1986 in the NSF Phase I backbone network, which used Fuzzball routers [46]. It is a tribute to the manufacturer that all four radios are serviceable today; two are in regular operation at University of Delaware, a third serves as backup spare and the fourth is in the Boston Computer Museum.

These four radios, together with a Heath WWV receiver at COMSAT Laboratories and a pair of TrueTime GOES satellite receivers at Ford Motor Headquarters and Digital Western Research Laboratories, provided primary time synchronization services throughout the ARPANET, MILNET and dozens of college campuses,

research institutions and military installations. By 1988 two Precision Standard Time WWV receivers joined the flock, but these along with the Heath WWV receiver are no longer available. From the early 1990s these nine primary servers were joined by an increasing number of volunteer radio-equipped servers now numbered over 120 in the public Internet.

As the cost of GPS receivers plummeted from the stratosphere (the first one this author bought cost $17,000), GPS receivers started popping up all over the place. In the US and Canada the longwave radio alternative to GPS is WWVB transmitting from Colorado, while in Europe it is DCF77 from Germany. However, shortwave radio WWV from Colorado, WWVH from Hawaii and CHU from Ontario have been useful sources. While GOES satellite receivers are available, GPS receivers are much less expensive and provide better accuracy. Over the years some 44 clock driver modules supporting these and virtually every radio, satellite and modem national standard time service in the world have been implemented for NTP.

Recent additions to the driver library include drivers for the WWV, WWVH and CHU transmissions that work directly from an ordinary shortwave receiver and audio sound card or motherboard codec. Some of the more exotic drivers built in our laboratory include a computerized LORAN-C receiver with exceptional stability [38] and a DSP-based WWV demodulator/decodor using theoretically optimal algorithms [25].

6. Hunting the Nanoseconds

When the Internet first wound NTP clocksprings, computers and networks were much, much slower than today. A typical WAN speed was 56 kb/s, about the speed of a telephone modem of today. A large timesharing computer of the day was the Digital Equipment TOPS-20, which wasn't a whole lot faster, but did run an awesome version of Zork. This was the heyday of the minicomputer, the most ubiquitous of which was the Digital Equipment PDP11 and its little brother the LSI-11. NTP was born on these machines and grew up with the Fuzzball operating system. There were about two dozen Fuzzballs scattered at Internet hotspots in the US and Europe. They functioned as hosts and gateways for network research and prototyping and so made good development platforms for NTP.

In the early days most computer hardware clocks were driven by the power grid as the primary timing source. Power grid clocks have a resolution of 16 or 20 ms, depending on country, and the uncorrected time can wander several seconds over the day and night, especially in summertime. While power grid clocks have rather dismal performance relative to accurate civil time, they do have an interesting characteristic, at least in areas of the country that are grid-synchronous. Early experiments in time synchronization and network measurement could assume the time offsets between grid-synchronized clocks was constant, since they all ran at the same frequency and close phase, so all NTP had to do was calibrate the constant offsets.

Later, computer clocks were driven by an oscillator stabilized by a quartz crystal resonator, which is much more stable than the power grid, but has the disadvantage that the intrinsic frequency offset between crystal clocks can reach several hundred parts-per-million (PPM) or several seconds per day. In fact, over the years only Digital has paid particular attention to the manufacturing tolerance of the clock oscillator, so their machines make the best timekeepers. In fact, this is one of the reasons why all the primary servers operated by NIST are Digital Alphas.

As crystal clocks came into widespread use, the NTP clock discipline algorithm was modified to adjust the frequency as well as the time. Thus, an intrinsic offset of several hundred PPM could be reduced to a residual in the order of 0.1 PPM and residual timekeeping errors to the order of a clock tick. Later designs decreased the tick from 16 or 20 ms to 4 ms and eventually to 1 ms in the Alpha. The Fuzzballs were equipped with a hardware counter/timer with 1-ms tick, which was considered heroic in those days.

To achieve resolutions better than one tick, some kind of auxiliary counter is required. Early Sun SPARC machines had a 1-MHz counter synchronized to the tick interrupt. In this design, the seconds are numbered by the tick interrupt and the microseconds within the second read directly from the counter. In principle, these machines could keep time to 1 μs, assuming that NTP could discipline the clocks between machines to this order. In point of fact, performance was limited to a few milliseconds, both because of network and operating system jitter and also because of small varying frequency excursions induced by ambient temperature variations.

Analysis, simulation and experiment led to continuing improvements in the NTP clock discipline algorithm, which adjusts the clock time and frequency in response to an external source, such as another NTP server or a local source such as a radio or satellite receiver or telephone modem [35]. As a practical matter, the best timekeeping requires a directly connected radio; however, the interconnection method, usually a serial port, itself

has inherent jitter. In addition, the method implemented in the operating system kernel to adjust the time generally has limitations of its own [41].

In a project originally sponsored by Digital and later by Sun, components of the NTP clock discipline algorithm were implemented directly in the kernel. In addition, an otherwise unused counter was harnessed to interpolate the microseconds. In addition to these improvements, a special clock discipline loop was implemented for the pulse-per-second (PPS) signal produced by some radio clocks and precision oscillators. The complete design and application interface was reported in [32], some sections of which appeared as RFC-1589 [34]. The kernel software is now an integral component of the kernels distributed with Digital and Sun workstations.

A systematic search for sources of jitter described in the paper [29] revealed significant contributions due to serial port drivers, I/O system latencies and process scheduling. Most of these latencies were avoided using crafted I/O appliques in the form of BSD line disciplines and STREAMS modules. Van Jacobson and Craig Leres at Lawrence Berkeley Laboratory, built one that used PPS signal transitions on a serial port lead to generate a precision timestamp with a latency of only 6 μs. However, these appliques were in general not portable and were implemented for only a few systems.

An interesting application of the PPS signal was in Norway, where a Fuzzball NTP primary server was connected to a cesium frequency standard with PPS output. In those days the Internet bridging the US and Europe had notoriously high jitter, in some cases peaks reaching over one second. The cesium standard and kernel discipline maintained constant frequency, but did not provide a way to number the seconds. NTP provided this function via the Internet and other primary servers. The experience with very high jitter resulted in special nonlinear signal processing code, called the popcorn spike suppressor, in the NTP clock discipline algorithm.

Still, network and computer speeds were reaching higher and higher. The time to cycle through the kernel and back, once 40 μs in a Sun SPARC IPC, was decreasing to 1 μs in a Digital Alpha and 0.4 μs in a Sun Blade 1000. In order to insure a reliable ordering of events, the need was building to improve the clock resolution better than 1 μs and the nanosecond seemed a good target. NTP Version 4 now implements all clock adjustments in floating double, which in principle could discipline the clock with femtosecond resolution, if the underlying hardware supported it.

For the ultimate accuracy, the original microsecond kernel was overhauled to support a nanosecond clock con-

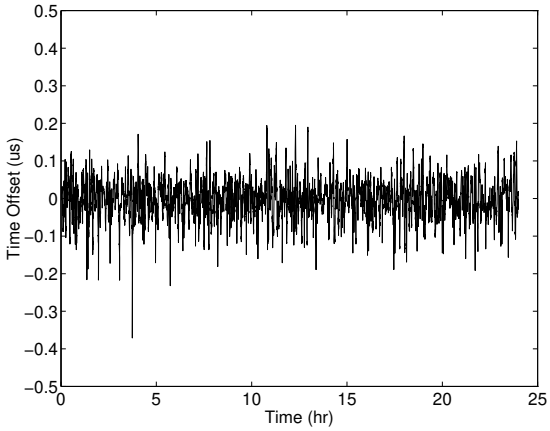

Figure 1. Nanokernel Clock Discipline with PPS Signal on Alpha 433au (from [20])

forming to the PPS interface specified in RFC-2783 [54]. Nanosecond kernels have been built and tested for SunOS, Alpha, Linux and FreeBSD systems, the latter two of which include the code in current systems [20]. The results with the new kernel demonstrate that the residual RMS error with modern hardware and a precision PPS signal is in the order of 50 ns [21]. The skeptic should see Figure 1, although admittedly this shows the jitter and not the systematic offset, which must be calibrated separately.

This represents the state of the art in current timekeeping practice. Having come this far, the machine used by this author now runs at 2GHz and can chime with another across the country at full 100-Mb/s speeds, which raises the possibility of a picosecond clock. The inherent resolution of the NTP timestamp is about 232 picoseconds, which suggests we soon might approach that limit and require rethinking the NTP protocol design. At these speeds NTP could be used to synchronize the motherboard CPU and ASIC oscillators using optical interconnects.

7. Experimental Studies

Over the years a good deal of effort has gone into the analysis of computer clocks and methods to stabilize them in frequency and time. As networks and computers have become faster and faster, the characterization of computer clock oscillators and the evolution of synchronization technology has continuously evolved to match. Following is a technical timeline on the significant events in this progress.

When the ICMP protocol divorced from the first Internet routing protocol GGP, one of the first functions added to ICMP was the ICMP Timestamp message, which is similar to the ICMP Echo message, but carries timestamps with millisecond resolution [59]. Experiments with

these messages used Fuzzballs and the very first implementation of ICMP. In fact, the first use of the name PING (Packet InterNet Groper) can be found in RFC-889 [51]. Related experiments were later reported by Cole [3].

While the hosts and gateways did not at first have synchronize clocks, they did record timestamps with a granularity of 16 ms or 1 ms, which could be used to measure roundtrip times and synchronize experiments after the fact. Statistics collected this way were used for the analysis and refinement of early TCP algorithms, especially the parameter estimation schemes used by the retransmission timeout algorithm.

The first comprehensive survey of NTP operating in the Internet was published in 1985 [48]. Later surveys appeared in 1990 [42] and 1997 [27]. The 1997 survey was a profound undertaking. It attempted to find and expose every NTP server and client in the public Internet using data collected by the standard NTP monitoring tools. After filtering to remove duplicates and falsetickers, the survey found over 185,000 client/server associations in over 38,000 NTP servers and clients. The results actually represented only a fraction of the total number of NTP servers and clients. It is known from other sources that many thousands of NTP servers and clients lurk behind firewalls where the monitoring programs can't find them. Extrapolating from data provided about the estimated population in Norway, it is a fair statement that well over 100,000 NTP daemons were prowling the Internet in 1997 and more likely several times that number. Recently, a NTP client was found hiding in a standalone print server. The next one may be found in an alarm clock.

The paper [39] is a slightly tongue-in-cheek survey of the timescale, calendar and metrology issues involved in computer network timekeeping. Of particular interest in that paper was how to deal with leap seconds in the UTC timescale. While provisions are available in NTP to disseminate leap seconds throughout the NTP subnet, means to anticipate their scheduled occurrence was not implemented in radio, satellite and modem services until relatively recently and not all radios and only a handful of kernels support leap seconds. If fact, on the eleven leap second occasions since NTP began in the Internet until 1997, the behavior of the NTP subnet on and shortly after each leap could only be described in terms of a pinball machine.

Today there is no excuse for leap second misadventures. In NTP Version 4 the leap bits are set automatically in the NIST and USNO servers and passed up the stratum tree by dependent servers and clients.

While almost all time dissemination means in the world are based on Coordinated Universal Time (UTC), some users have expressed the need for TAI, including means to metricate intervals that span multiple leap seconds [11]. NTP Version 4 includes a simple mechanism to retrieve a table of historic leap seconds from NIST servers and distribute it throughout the NTP subnet. However, at this writing a suitable API has yet to be designed and implemented and then navigate the IETF standards process. Refinements to the Autokey protocol have been made to insure the most recentcopy of this table is distributed using secure means.

8. Theory and Algorithms

As all this was going on, there was a good deal of excitement in the theoretical community developing abstract models and algorithms for time synchronization. The fundamental abstraction from which correctness principles are based is the *happens-before* relation introduced by Lamport [9]. Lamport, et al, [10] show that $3m + 1$ clocks are required in order to determine a reliable time value if no more than m of them are falsetickers, but only $2m + 1$ clocks are required if digital signatures are used, as is the case with NTP Autokey.

Interactive-consistency algorithms use a Byzantine agreement protocol involving successive rounds of readings, possibly relayed and possibly augmented by digital signatures. Examples include the fireworks algorithm of Halpern, et al, [8] and the optimum algorithm of Srikanth and Toueg [65]. These algorithms require large numbers of messages and are designed to detect faults that have rarely been found in the Internet experience. Convergence algorithms use statistical clustering techniques such as the FTA and CNV algorithms of Lundelius and Lynch [17], the majority-subset algorithm of Mills [49], the non-Byzantine algorithm of Rickert [61] and the egocentric algorithm of Schneider [64].

The particular choice of offset measurement and computation procedure used in NTP is a variant of the returnable-time system used in some digital telephone networks as described by Lindsay and Kantak [14]. The clock filter and selection algorithms are designed so that the clock synchronization subnet self-organizes as a hierarchical-master-slave configuration as in Mitra [55]. A different approach is the probabilistic scheme suggested by Cristian [4] and implemented in a distributed context by Arvind [2] and Liao [13]. For various reasons, none of these algorithms in their original form perform well with the extreme jitter prevalent in the Internet.

Drawing from this work, a cascade of four algorithms was developed for NTP, including the filter, selection, clustering and combining algorithms. In a series of experiments documented in RFC-956 [48], the algorithm that emerged computes the roundtrip delay for each of the last eight measurement rounds and selects the offset associated with the minimum roundtrip delay. This algorithm, somewhat modified, survives to the present.

The NTP selection algorithm is based on the intersection algorithm of Marzullo and Owicki [18], together with a refinement algorithm similar to the self-stabilizing algorithm of Lu [16]. The maximum error bound for any server is represented by an interval equal to the roundtrip delay with center the apparent time. A clique is formed from a number of servers whose intervals overlap. If there is a clique containing a majority of servers, those servers are *truechimers*. All the rest are *falstickers*. If no clique has a majority of servers, no decision is possible.

The truechimers which survive of the intersection algorithm are processed by the clustering algorithm, which repeatedly casts out outlyers furthest from the cluster median until a minimum number of survivors remain. While the details differ, this algorithm is fundamentally the same as used by Paxton [57]. Finally, the surviving offsets are combined using a weighted average to form the final offset used to discipline the computer clock, which is essentially the same technique NIST uses to wrangle their herd of cesium clocks [1].

The fundamentals of computer clock discipline technology were presented in the 1992 report [36], which remains valid today. That report set forth mathematically precise models for error analysis, transient response and clock discipline principles. Selected sections of that report were condensed and refined in the paper [35].

In a series of careful measurements over a period of two years with selected servers in the US, Australia and Europe, an analytical model of the idiosyncratic computer clock oscillator was developed and verified. While a considerable body of work on this subject has accreted in the literature, with few exceptions the object of study has been precision oscillators of the highest quality used as time and frequency standards. Computer oscillators have no such pedigree, since there are generally no provisions to stabilize the ambient environment, in particular the crystal temperature.

The clock discipline algorithm in the paper [31] further extended and refined the model evolved from the report

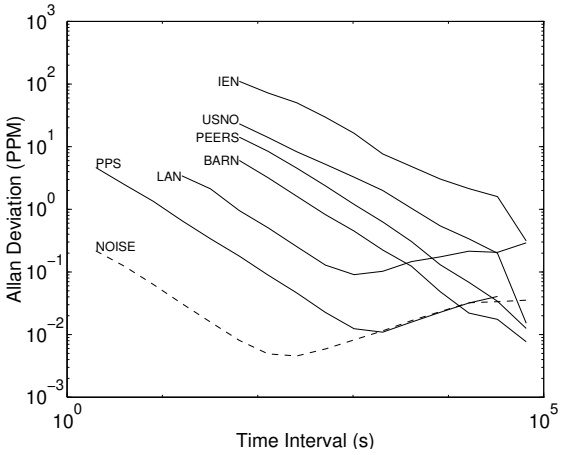

Figure 2. Allan Deviation Plot (from [24])

[36]. The algorithm design was considerably influenced by a collaboration with Judah Levine at NIST. Levine's own *lockclock* algorithm, which is used in the NTP primary servers operated by NIST, is described in his paper [12].

The paper [31] introduced the concept of *Allan deviation*, a statistic useful for the characterization of clock discipline performance. This statistic is commonly displayed by a plot of stability versus averaging interval in log-log coordinates, as shown in Figure 2. Each Internet server is characterized by a straight line with slope −1, which is associated with white phase noise. Each computer oscillator is characterized by a straight line with slop +0.5, which is associated with random-walk frequency noise. Knowing these two characteristics allows the optimum averaging interval to be determined for each combination of server and oscillator. The paper also described a number of algorithmic improvements incorporated in the NTP Version 4 design, including the intersection algorithm and an improved discipline and hardware model for the kernel PPS signal.

The paper [24] further extended and quantified the clock discipline model developed in [31]. Its primary contribution is the *Allan intercept* model which characterizes typical computer oscillators. The Allan intercept is the point (x, y) where the straight-line asymptotes shown in Figure 2 for each individual source and oscillator intersect. This work resulted in a hybrid algorithm, implemented in NTP Version 4, which both improves performance over typical Internet paths and allows the message poll intervals to be substantially increased without degrading accuracy. A special purpose simulator including substantially all the NTP algorithms was used to verify predicted behavior with both simulated and actual data over the entire envelope of frenetic Internet behaviors.

9. Growing Pains

In the beginning, almost all NTP servers operated in client/server mode, where a client sends requests at intervals ranging from one minute to tens of minutes, depending on accuracy requirements. In this mode, time values flow outward from the primary servers through possibly several layers of secondary servers to the clients. In some cases involving multiply redundant servers, peers operate in symmetric mode and values can flow from one peer to the other or vice versa, depending on which one is closest to the primary source according to a defined metric. Some institutions like University of Delaware and GTE, for example, operate multiple primary servers, each connected to one or more redundant radio and satellite receivers using different dissemination services. This forms an exceptionally robust synchronization source for both on-campus and off-campus public access.

While NTP service makes only minimal demands on the host processor, the client population of popular servers like NIST and USNO has been growing to huge proportions, with the busiest servers handling several hundred packets per second. In the Internet of today, even this flux doesn't daunt these servers, at least if the clients use common sense and reasonable message poll intervals.

However, common sense is not a ubiquitous commodity even in the NTP community. Whether due to terror or ignorance, on occasion clients send packets as fast as possible, like 256 packets in one second. In Australia, where packets are counted and charged, this is an unacceptable hazard.

A suite of defensive measures has been incorporated in the latest NTP design. One of these, called the *kiss-of-death packet*, is returned to a misbehaving client sending at terrorist rates. If the client uses the public NTP distribution, receiving this packet causes the client association to be demobilized and a pique sent to the system log.

Not every NTP implementation is as polite as the public distribution. In order to defend against clogging attacks like the 256-packet miscreant, a feature similar to what telephone providers defube *call-gap* has been incorporated in the design. The scheme uses a LRU list of IP source addresses and time values and operates to discard packets which exceed given peak and average rate limits.

10. As Time Goes By

In retrospect, while NTP has been a technical adventure in its own right by providing the means for accurate and dependable time synchronization, NTP has also been an enabling technology for practical uses of synchronized clocks. Using NTP for timestamping stock trades, radio and television broadcast programs and distributed data acquisition readily springs to mind; but, as Liskov points out [15], synchronized clocks are vital to some important distributed algorithms and could improve performance in others.

At the beginning of the new century it is quite likely that precision timekeeping technology has evolved about as far as it can given the realities of available computer hardware and operating systems. Using specially modified kernels and available interface devices, Poul-Henning Kamp and this author have demonstrated that computer time in a modern workstation can be disciplined within some tens of nanoseconds relative to a precision source such as a cesium or rubidium frequency standard [21]. While not many computer applications would justify such heroic means, the demonstration suggests that the single most useful option for high performance timekeeping in a modern workstation may be a temperature compensated or stabilized oscillator.

It is likely that future deployment of public NTP services might well involve an optional secure timestamping service, perhaps for-fee. This agenda is being pursued in a partnership with NIST and Certified Time, Inc. In fact, several NIST servers are now being equipped with timestamping services. This makes public-key authentication a vital component of such a service, especially if the Sun never sets on the service area.

11. Acknowledgements

Internet timekeeping is considered by some to be a hobby, and even this author has revealed a likeness to amateur radio. There seems no other explanation why the volunteer timekeeper corps has continued so long to improve the software quality, write clock drivers for every new radio that comes along and port the stuff to new hardware and operating systems. The generals in the army have been revealed in the narrative here, but the many soldiers of the trench must be thanked as well, especially when the hardest job is convincing the boss that time tinkering is good for business. A list of important contributors is available at www.ntp.org and in the copyright page in the NTP software documentation.

12. References

Note: The papers and reports by D. L. Mills are available in PostScript and PDF at www.eecis.udel.edu/~mills.

1. Allan, D.W., J.E. Gray and H.E. Machlan. The National Bureau of Standards atomic time scale: generation, stability, accuracy and accessibility. In: Blair, B.E. (Ed.). *Time and Frequency Theory and Fundamentals*. National Bureau of Standards Monograph 140, U.S. Department of Commerce, 1974, 205-231.

2. Arvind, K. Probabilistic clock synchronization in distributed systems. *IEEE Trans. on Parallel and Distributed Systems 5, 5* (May 1964), 474-487.

3. Cole, R., and C. Foxcroft. An experiment in clock synchronisation. *The Computer Journal 31, 6* (1988), 496-502.

4. Cristian, F. A probabilistic approach to distributed clock synchronization. *Distributed Computing 3* (1989), 146-158.

5. Crocroft, J., and J.P. Onions. Network Time Protocol (NTP) Over the OSI Remote Operations Service. Network Working Group Report RFC-1165, University College London, June 1990, 10 pp.

6. Digital Time Service Functional Specification Version T.1.0.5. Digital Equipment Corporation, 1989.

7. Gusella, R., and S. Zatti. TEMPO - A network time controller for a distributed Berkeley UNIX system. *IEEE Distributed Processing Technical Committee Newsletter 6, NoSI-2* (June 1984), 7-15. Also in: *Proc. Summer 1984 USENIX* (Salt Lake City, June 1984).

8. Halpern, J.Y., B. Simons, R. Strong and D. Dolev. Fault-tolerant clock synchronization. *Proc. ACM Third Annual Symposium on Principles of Distributed Computing* (August 1984), 89-102.

9. Lamport, L., Time, clocks and the ordering of events in a distributed system. *Comm. ACM 21, 7* (July 1978), 558-565.

10. Lamport, L., and P.M. Melliar-Smith. Synchronizing clocks in the presence of faults. *JACM 32, 1* (January 1985), 52-78.

11. Levine, J., and D. Mills. Using the Network Time Protocol to transmit International Atomic Time (TAI). *Proc. Precision Time and Time Interval (PTTI) Applications and Planning Meeting* (Reston VA, November 2000), 431-439.

12. Levine, J. An algorithm to synchronize the time of a computer to universal time. *IEEE/ACM Trans. on Networking 3, 1* (February 1995), 42-50.

13. Liao, C., M. Martonosi, and D. Clark. Experience with an adaptive globally-synchronizing clock algorithm. *Proc. 11th Annual ACM Simposium on Parallel Algorithms and Architecture* (Saint Malo, June 1999), 106-114.

14. Lindsay, W.C., and A.V. Kantak. Network synchronization of random signals. *IEEE Trans. Communications COM-28, 8* (August 1980), 1260-1266.

15. Liskov, B. Practical uses of synchronized clocks in distributed systems. *Proc. 10th Annual ACM Symposium on Principles of Distributed Computing* (Montreal, April 1991), 1-9.

16. Lu, M., D. Zhang. Analysis of self-stabilizing clock synchronization by means of stochastic Petri nets. *IEEE Trans. Computers 39, 5* (May 1990), 597-604.

17. Lundelius, J., and N.A. Lynch. A new fault-tolerant algorithm for clock synchronization. *Proc. Third Annual ACM Symposium on Principles of Distributed Computing* (August 1984), 75-88.

18. Marzullo, K., and S. Owicki. Mantaining the time in a distributed system. *ACM Operating System Review 19, 3* (July 1985), 44-54.

19. Mills, D.L. The Autokey security architecture, protocol and algorithms. Electrical Engineering Report 03-2-20, University of Delaware, February 2003. 50 pp.

20. Mills, D.L. The nanokernel. Software and documentation, including test results, at www.ntp.org.

21. Mills, D.L., and P.-H. Kamp. The nanokernel. *Proc. Precision Time and Time Interval (PTTI) Applications and Planning Meeting* (Reston VA, November 2000), 423-430.

22. Mills, D.L. Public key cryptography for the Network Time Protocol. Electrical Engineering Report 00-5-1, University of Delaware, May 2000. 23 pp.

23. Mills, D.L. Cryptographic authentication for real-time network protocols. In: *AMS DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol. 45* (1999), 135-144.

24. Mills, D.L. Adaptive hybrid clock discipline algorithm for the Network Time Protocol. *IEEE/ACM Trans. Networking 6, 5* (October 1998), 505-514.

25. Mills, D.L. A precision radio clock for WWV transmissions. Electrical Engineering Report 97-8-1, University of Delaware, August 1997, 25 pp.

26. Mills, D.L. Clock discipline algorithms for the Network Time Protocol Version 4. Electrical Engineering Report 97-3-3, University of Delaware, March 1997, 35 pp.

27. Mills, D.L., A. Thyagarajan and B.C. Huffman. Internet timekeeping around the globe. *Proc. Precision Time and Time Interval (PTTI) Applications and Planning Meeting* (Long Beach CA, December 1997), 365-371.

28. Mills, D.L. Simple network time protocol (SNTP) version 4 for IPv4, IPv6 and OSI. Network Working Group Report RFC-2030, University of Delaware, October 1996, 18 pp.

29. Mills, D.L. The network computer as precision timekeeper. *Proc. Precision Time and Time Interval (PTTI) Applications and Planning Meeting* (Reston VA, December 1996), 96-108.

30. Mills, D.L. Proposed authentication enhancements for the Network Time Protocol version 4. Electrical Engineering Report 96-10-3, University of Delaware, October 1996, 36 pp.

31. Mills, D.L. Improved algorithms for synchronizing computer network clocks. *IEEE/ACM Trans. Networks 3, 3* (June 1995), 245-254.

32. Mills, D.L. Unix kernel modifications for precision time synchronization. Electrical Engineering Department Report 94-10-1, University of Delaware, October 1994, 24 pp.

33. Mills, D.L, and A. Thyagarajan. Network time protocol version 4 proposed changes. Electrical Engineering Department Report 94-10-2, University of Delaware, October 1994, 32 pp.

34. Mills, D.L. A kernel model for precision timekeeping. Network Working Group Report RFC-1589, University of Delaware, March 1994. 31 pp.

35. Mills, D.L. Precision synchronization of computer network clocks. *ACM Computer Communication Review 24, 2* (April 1994). 28-43.

36. Mills, D.L. Modelling and analysis of computer network clocks. Electrical Engineering Department Report 92-5-2, University of Delaware, May 1992, 29 pp.

37. Mills, D.L. Network Time Protocol (Version 3) specification, implementation and analysis. Network Working Group Report RFC-1305, University of Delaware, March 1992, 113 pp.

38. Mills, D.L. A computer-controlled LORAN-C receiver for precision timekeeping. Electrical Engineering Department Report 92-3-1, University of Delaware, March 1992, 63 pp.

39. Mills, D.L. On the chronology and metrology of computer network timescales and their application to the Network Time Protocol. *ACM Computer Communications Review 21, 5* (October 1991), 8-17.

40. Mills, D.L. Internet time synchronization: the Network Time Protocol. *IEEE Trans. Communications COM-39, 10* (October 1991), 1482-1493.

41. Mills, D.L. On the accuracy and stability of clocks synchronized by the Network Time Protocol in the Internet system. *ACM Computer Communication Review 20, 1* (January 1990), 65-75.

42. Mills, D.L. Measured performance of the Network Time Protocol in the Internet system. Network Working Group Report RFC-1128. University of Delaware, October 1989, 18 pp.

43. Mills, D.L. Network Time Protocol (Version 2) specification and implementation. Network Working Group Report RFC-1119, 61 pp. University October 1989, 27 pp.

44. Mills, D.L. The Fuzzball. *Proc. ACM SIGCOMM 88 Symposium* (Palo Alto CA, August 1988), 115-122.

45. Mills, D.L. Network Time Protocol (Version 1) specification and implementation. Network Working Group Report RFC-1059. University of Delaware, July 1988.

46. Mills, D.L., and H.-W. Braun. The NSFNET Backbone Network. *Proc. ACM SIGCOMM 87 Symposium* (Stoweflake VT, August 1987), 191-196.

47. Mills, D.L. Network Time Protocol (NTP). Network Working Group Report RFC-958, M/A-COM Linkabit, September 1985.

48. Mills, D.L. Experiments in network clock synchronization. Network Working Group Report RFC-957, M/A-COM Linkabit, September 1985.

49. Mills, D.L. Algorithms for synchronizing network clocks. Network Working Group Report RFC-956, M/A-COM Linkabit, September 1985.

50. Mills, D.L. DCN local-network protocols. Network Working Group Report RFC-891, M/A-COM Linkabit, December 1983.

51. Mills, D.L. Internet delay experiments. Network Working Group Report RFC-889, M/A-COM Linkabit, December 1983.

52. Mills, D.L. DCNET internet clock service. Network Working Group Report RFC-778, COMSAT Laboratories, April 1981.

53. Mills, D.L. Time synchronization in DCNET hosts. Internet Project Report IEN-173, COMSAT Laboratories, February 1981.

54. Mogul, J., D. Mills, J. Brittenson, J. Stone and U. Windl. Pulse-per-second API for Unix-like operating systems, version 1. Request for Comments RFC-2783, Internet Engineering Task Force, March 2000, 31 pp.

55. Mitra, D. Network synchronization: analysis of a hybrid of master-slave and mutual synchronization. *IEEE Trans. Communications COM-28, 8* (August 1980), 1245-1259.

56. Mu, Y., and V. Varadharajan. Robust and secure broadcasting. *Proc. INDOCRYPT 2001, LNCS 2247*, Springer Verlag, 2001, 223-231.

57. Paxson, V. On calibrating measurements of packet transit times. *Proc. Joint Internet Conference on Measurements and Modelling of Computer Systems* (Madison, June 1998), 11-21.

58. Postel, J. Daytime Protocol. Network Working Group RFC-867, Information Sciences Institute, May 1983, 2 pp.

59. Postel, J. Internet control message protocol. Network Working Group RFC-777. Information Sciences Institute, April 1981, 14 pp.

60. Postel, J., and K. Herrenstien. Time Protocol. Network Working Group RFC-868. Information Sciences Institute, May 1983, 2 pp.

61. Rickert, N.W. Non Byzantine clock synchronization - a programming experiment. *ACM Operating Systems Review 22, 1* (January 1988), 73-78.

62. Schneider, F. A paradigm for reliable clock synchronization. Computer Science Technical Report TR 86-735, Cornell University, February 1986, 19 pp.

63. Schnorr, C.P. Efficient signature generation for smart cards. *J. Cryptology 4, 3* (1991), 161-174.

64. Schneider, F. A paradigm for reliable clock synchronization. Computer Science Technical Report TR 86-735, Cornell University, February 1986, 19 pp.

65. Srikanth, T.K., and S. Toueg. Optimal clock synchronization. *JACM 34, 3* (July 1987), 626-645.

66. Su, Z. Specification of the Internet Protocol (IP) timestamp option. Network Working Group RFC-781, May 1981, 2 pp.