# Time Surveying: Clock Synchronization over Packet Networks

by

**Gregory D. Troxel**

S.B., Massachusetts Institute of Technology
(1987)

S.M., E.E., Massachusetts Institute of Technology
(1990)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degrees of

**Doctor of Philosophy in**
**Electrical Engineering and Computer Science**

at the

**Massachusetts Institute of Technology**

May, 1994



Signature of Author ________________________________________
Department of Electrical Engineering and Computer Science
May 12, 1994

Certified by ________________________________________
David Clark
Thesis Supervisor

Accepted by ________________________________________
Frederic R. Morgenthaler
Chairman, Departmental Committee on Graduate Students

**Time Surveying: Clock Synchronization over Packet Networks**

by

**Gregory D. Troxel**

Submitted to the Department of Electrical Engineering and Computer Science on May 12, 1994 in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Electrical Engineering and Computer Science

## Abstract

This thesis applies the methods and mindset of a high-accuracy land surveyor to the problem of clock synchronization over packet networks. It presents a novel technique for analyzing synchronization performance, termed the *running sum of adjustments*, or `rsadj`, synchronization measurement technique. The new technique does not require the use of an accurate local time reference. It presents a novel synchronization scheme, based on the `rsadj` technique, termed integrated timekeeping. The thesis discusses what hardware support is necessary to support precision timekeeping, and argues that only very simple mechanisms are needed.

The `rsadj` synchronization measurement technique completely decouples the effects of the synchronization algorithm from the underlying behavior of the computer's oscillator, the network and remote clocks. The measurement technique uses the oscillator of the computer as an important measurement tool; it simultaneously characterizes the oscillator and uses it to analyze the behavior of remote clocks. It estimates oscillator parameters, and computes how the clock *should have been steered.* The original data and what the behavior would have been had the clock been optimally steered are examined. The thesis describes an implementation of the technique, and examines actual synchronization behavior. The thesis examines the behavior of an important and widely deployed synchronization mechanism, the Network Time Protocol (NTP). It shows that NTP does not fully exploit the stability of the computer's oscillator under some conditions, particularly when the oscillator is very stable (a few parts in $10^8$) and significant numbers of packets encounter 10 ms or more of additional delay. The thesis suggests incremental changes to improve NTP.

The thesis presents a novel technique for synchronization, termed *integrated timekeeping*, which periodically estimates parameters of the computer's oscillator and sets the system clock based on the estimated parameters. The thesis presents data showing the behavior of clocks synchronized by NTP and by the integrated timekeeping scheme, obtained by simulation and by an implementation. Given realistic network conditions and a local oscillator stable to a few parts in $10^8$, the new scheme achieves timekeeping stability of approximately an order of magnitude better than NTP.

Keywords: oscillator, Internet, NTP

| | |
|---|---|
| Thesis Supervisor: | David Clark |
| Title: | Principal Research Scientist |

## Acknowledgments

I would like to thank David Mills for his contagious devotion to accurate time, and for his authorship of the Network Time Protocol, without which this work would almost certainly not have occurred. I have also appreciated his encouragement over the years to try to actually synchronize clocks. I appreciate very much his loan of a WWVB receiver to my research group, and his continuing efforts to provide a high-quality implementation of NTP to the Internet community.

I would like to thank Ty Sealy, Tom Greene, Charles Robinson, William Ang, and Kevin Garceau of the MIT Laboratory for Computer Science for their assistance in installing experimental apparatus for this research, securing permission to install equipment in space not controlled by the Laboratory, and helping with the inevitable problems that occur with such installations. I would like to thank the staff of BTE Engineering for their acceptance of my equipment in their lunchroom.

I would like to thank Bill Chiarchiaro for the loan of one of the oscillators used in this research. John Paul Braud helped with the derivation of formulas used in the very early days of this work. Karen Palmer and other current and former residents of Ashdown House helped along the way in ways that might seem small but were important and much appreciated.

Many members of the Advanced Network Architecture and Telemedia, Networks, and Systems research groups of the MIT Laboratory for Computer Science were helpful during this research. I would like to thank Mike Ciholas for the many hours of helpful discussions about hardware requirements for synchronization, and his work in the design and construction of the prototype GDT-TIME hardware timekeeping module. I had many useful discussions with Tim Shepard, who was always willing to listen to me attempt to explain my schemes, and provided helpful comments on drafts of this document. He also helped greatly in taking care of experimental apparatus at times when it needed attention and I was not at the lab. His authorship of the plotting program used in this work is greatly appreciated, as are contributions to that program made by Andrew Heybey. Vanu Bose read drafts of several chapters, and provided many helpful comments. I appreciate the typographical assistance of Mark Reinhold, who rescued me from having incompatible fonts in the figures. I appreciate the assistance of Kathi-Jo Madera and Lisa Taylor during the several years over which this research was conducted.

I would like to thank my thesis supervisor, David Clark, for encouraging me to pursue this research even though it was not directly relevant to the research agenda of his research group. Our many discussions helped greatly in refining the ideas presented in this document from their original muddled form into ones that can be stated clearly and precisely.

I would like to thank Tom Herring, who served as a reader for this thesis, for his helpful comments on drafts of the document. As a student in his class on the Global Positioning System I became interested in surveying, and this led to a greater understanding of the problem at hand. I would like to thank Barbara Liskov for her extremely helpful comments, particularly on the presentation of material in this document, and for having shown that synchronizing clocks well almost all of the time is very useful.

I would like to thank my parents for their encouragement to attend graduate school, their support over many years, and for instilling in me a desire to learn. My sisters, Andrea and Jocelyn, provided statistical advice and encouragement. I would like to thank my wife, Mary Ellen, for her support, love, and companionship. Her constant encouragement made writing the document much more pleasant than it otherwise would have been, and her expert proofreading of final drafts helped greatly.

This research was supported by the Advanced Research Projects Agency under contract number DABT63–92–C–0002.

# Contents

# Chapter 1
# Introduction

In this thesis I apply the methods and mindset of a high-accuracy land surveyor to the problem of clock synchronization over packet networks. I present novel approaches to analyzing synchronization behavior and to synchronizing clocks. I argue that only simple hardware support is necessary for precision timekeeping.

The problem of clock synchronization over packet networks is to cause a computer to have the correct time by exchanging messages with other computers. Generally, the computer to be synchronized does not initially have the correct time. In addition, its clock will generally not run at the correct frequency, so that setting its value of time correctly once does not solve the problem. I concentrate on causing a computer's time to be correct, rather than on setting the time on two separate computers to be the same. Because of this emphasis, I assume that specific other computers reachable via the network will have the correct time, or at least a closer approximation to the correct time than the local computer. This emphasis and assumption correspond to the problem of synchronizing many computers to the correct time as determined by national standards bodies. Mechanisms such as radio timing receivers exist to transfer correct time to a small number of computers. Other computers may then set their clocks by executing a synchronization algorithm which exchanges messages with computers closer to the standards organization in the synchronization network.

## 1.1 Overview

I present a novel technique for analyzing the behavior of any synchronization algorithm based on the use of offset measurements over packet networks, termed the `rsadj` technique, short for "running sum of adjustments." Rather than viewing a clock as something which is often incorrect and should be frequently adjusted, I treat the *local oscillator* of a computer, the device which indicates the passage of time, as distinct from the *system clock*, the object which indicates the current time. Given this distinction, I can obtain views of the behavior of remote clocks as seen by the local oscillator. Such a view is fundamentally different from the view of the remote clock relative to the system clock because the view relative to the local oscillator is uncontaminated by the actions of any synchronization algorithm. Also, I can obtain a view of the system clock relative to the local oscillator; this is precisely the behavior of the synchronization algorithm.

After distinguishing between the local oscillator and the system clock, I am able to separate the behavior of a synchronization scheme from that of the local oscillator, the packet network, and remote clocks to which observations are made. By observing multiple remote clocks, I can determine the stability of the local oscillator, and separate the behavior of the local oscillator from that of the network and remote clocks. By understanding the effects of variation in delay encountered by packets sent over the network on the basic offset measurement mechanism, I can often separate the behaviors of the network and remote clocks. I present and interpret data showing the behavior of local oscillators, networks, and

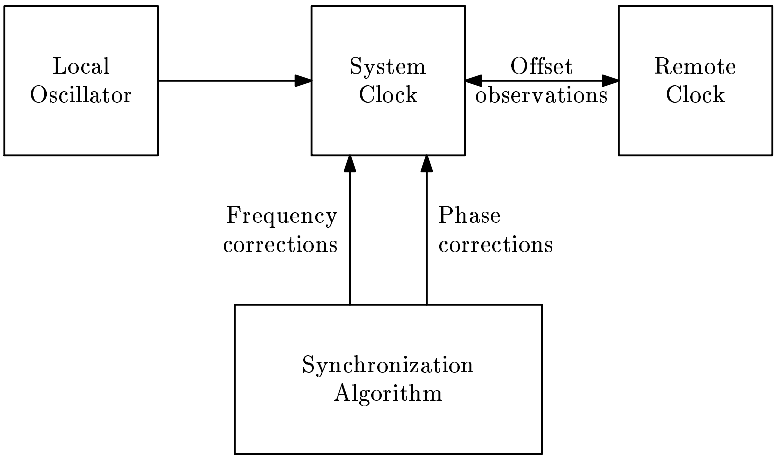



Figure 1.1: Relationship of local oscillator, system clock, synchronization algorithm, network, and remote clocks. The network is symbolized by the arrow between the system clock and remote clocks.

remote clocks. I argue that in a strict sense the task of fully separating network and remote clock behavior is fundamentally impossible, but also show that significant progress towards this goal can be achieved in real situations.

I use the `rsadj` measurement technique to observe the behavior of an important existing synchronization algorithm, the Network Time Protocol (NTP). I show that in situations where the local oscillator is stable to a few parts in $10^8$ and there is significant variation in delays encountered in the network, NTP fails to take advantage of the stability of the local oscillator, and that errors in offset observations due to delay variance impede highly stable synchronization. I suggest incremental improvements to NTP to ameliorate this problem.

After showing that the behaviors of the local oscillator, system clock, network, and remote clocks are separable, I describe a novel technique to synchronize clocks based on the methodology used to separate these behaviors. The new synchronization scheme is termed *integrated timekeeping* because it views the characteristics of the local oscillator, network and remote clocks as unknown. It both characterizes the local oscillator, and uses it as a measurement tool to decide which observations to use. Then, the local oscillator and estimates of its behavior are used to compute the value of the system clock.

I describe the details of the new synchronization scheme, and present data from a simulation of the scheme and from an implementation. I analyze the performance of integrated timekeeping both using the new measurement technique and by using a Global Positioning System (GPS) receiver as a source of "true time."

I discuss the requirements for hardware clocks in order to support precision timekeeping. Many computers are supplied with clock hardware that is totally inadequate for precision timekeeping. While a high-stability local oscillator is necessary for extremely precise timekeeping, the additional hardware support required is extremely simple, and could be provided at close to no cost by designers of computer systems.

I give more precise definitions of several terms used throughout this thesis. Figure 1.1 shows the relationships among the objects described by the various terms.

**local oscillator** device which generates ticks at approximately equal intervals in time, and an associated counter which contains a representation of time and is incremented by the nominal value of a tick when a tick occurs.

**system clock** logical object which contains a representation of the current time, generated by adding the nominal time interval of a tick when each tick occurs. It may also be adjusted by a synchronization algorithm in order to bring its value into agreement with the correct time.

**remote clock** entity which has a value of time, can normally be reached via a packet network, and supports a protocol for making a measurement of the difference in the value of its time and the local time.

**reference clock** entity which has a value of time, and is directly attached in some fashion to a computer. It must support a mechanism for making a measurement of the difference in value of its time and the local time. In many ways a reference clock is similar to a remote clock, except that it is of course not remote. Examples are the Global Positioning System (GPS) receiver I use later in this work, and various other receivers of timing signals.

**offset measurement** the process of sending a packet to a remote clock and back in order to obtain an estimate of the difference between the the time of remote clock and that of the system clock.

**synchronization algorithm** procedure that may observe the values of the local oscillator and the system clock and the results of all offset measurements. It may modify the value of the system clock and control the making of offset measurements.

## 1.2 What is clock synchronization, and why is it important?

David Mills has defined "clock synchronization" as bringing both the frequency of two clocks as well as their values of time at a particular epoch into agreement ([Mil92a], p. 2). I will adopt this definition. When using the term "clock synchronization" in reference to only one clock, the other "clock" implicitly involved in the synchronization will be "correct time," meaning Coordinated Universal Time (UTC) provided by the National Institute of Standards and Technology (NIST). This thesis focuses on synchronizing the value of a clock to the correct time by making offset observations to remote clocks presumed to have the correct time.

Let $\texttt{time}(t)$ be the value of the system clock of a computer at time $t$. By synchronizing time, I mean that the value of the system clock is always correct:

$$\forall t : \texttt{time}(t) = t$$

By synchronizing frequency, I mean that the difference between readings of the system clock at times $\tau$ apart is in fact $\tau$:

$$\forall t\ \forall \tau : \texttt{time}(t + \tau) - \texttt{time}(t) = \tau$$

Some computer applications depend on accurate timekeeping. For example the program "make" when accessing files via the Network File System (NFS) can fail if the clocks on the machine running "make" and the remote file server do not agree sufficiently closely. "Make" compares timestamps on files (generated by the file server) to the current time to determine whether the file must be regenerated from sources. Some algorithms do not depend on correct clocks, but are more efficient given accurate synchronization. Liskov, Shrira, and Wroclawski ([LSW91]) present an algorithm that is always correct and is efficient given synchronized clocks. Algorithms having this property benefit greatly from synchronization methods such as the integrated timekeeping scheme in which average performance is far better than the provable performance.

Even if the phase of the system clock cannot be set extremely accurately, accurately controlling its frequency can be useful. Consider a computer-controlled physics experiment in which a computer records information about events and the times at which they occur. It may be the case that a 5 ms error in the absolute time to which the events are recorded is unimportant, but an error in measuring the time between two events of 5 ms is undesirable. Or, consider a computer with a clock driven by a very stable oscillator whose exact frequency is unknown. Determining the frequency of this oscillator could allow it to be used to measure the frequencies of other signals in the laboratory more cheaply than purchasing a frequency standard with the required accuracy.

### 1.3 The Network Time Protocol

The Network Time Protocol (NTP) is a network protocol and a collection of algorithms for synchronizing computer clocks over packet networks. My work in some ways builds on prior work done in the development of NTP, and some parts of it would be difficult to understand without a general understanding of the workings of NTP. A summary of NTP follows, with emphasis on those aspects relevant to the current discussion. Further information can be found in the NTP Version 3 Specification ([Mil92b]).

NTP functions by repeatedly making offset observations to a number of remote clocks. It adjusts the value of the system clock periodically in order to bring its value closer to the correct time. NTP assumes that the remote clocks observed have the correct time.

The Network Time Protocol specifies the format of packets sent over the network, and the manner in which a computer conforming to the protocol must respond to received packets. In addition, the protocol specification includes suggested algorithms for managing the value of the local clock; an implementation may utilize other algorithms and still be compliant with the protocol. Thus, such algorithms are technically not part of the NTP specification.

The scheme to synchronize the local clock presented in Chapter 5 can be considered an alternative to the suggested algorithms, and an implementation of it could be NTP compliant. However, it would not be reasonable to refer to such an implementation as an "implementation of NTP," as that implies that the suggested algorithms were used. I consider both the specification of packet formats and responses and the suggested algorithms to be part of the definition of NTP.

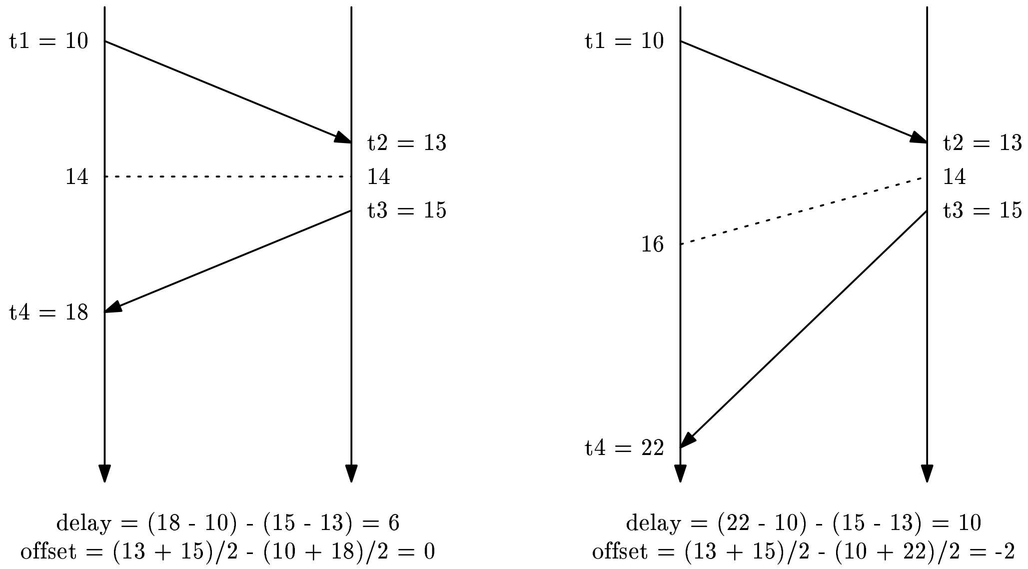



Figure 1.2: NTP offset measurements, with all times given in seconds. On the left is a nominal measurement, with equal delays. On the right is a measurement with asymmetric delays — the return trip encountered an additional 4 s of delay. These diagrams are inspired by similar ones in [Mil92b], p. 25 and p. 100.

### 1.3.1 Offset measurements

NTP makes observations of the offset between the system clock and the clock of a remote system ([Mil92b], pp. 24–26). A timestamp (called the "originate" timestamp) is taken and placed in a packet. The packet is sent to the remote system, where it is timestamped on receive (the "receive" timestamp), processed, timestamped on transmit (the "transmit" timestamp), and sent back. A timestamp is again taken when the packet is received at the system making the measurement. From the four timestamps the round-trip delay can be computed. For example, assume that the originate timestamp in the packet is 10 s, the receive timestamp 13 s, the transmit timestamp 15 s, and last timestamp 18 s, as shown in the left graph of Figure 1.2. The total time the packet was in transit and being processed at the remote host is the last timestamp minus the originate timestamp, or $18 - 10 = 8$ s. The time the packet was at the remote host is the transmit timestamp minus the receive timestamp, or $15 - 13 = 2$ s. Thus, the total delay attributed to the network is $8 - 2 = 6$ s.

The assumption is made that the half of the delay is attributable to each direction of packet travel, and the offset between the two clocks is computed. So, assuming the delay for each half of the packet's journey to be 3 s, we calculate that the remote computer's clock would have read $10 + 3 = 13$ s on receipt of the packet if the clocks were synchronized. It in fact had this value; in the example the clocks are already synchronized. Had the remote clock been 1 s fast relative to the local clock, the second and third timestamps would have been 14 s and 16 s, resulting in the same observed delay and an observed offset of 1 s.

The right graph of Figure 1.2 shows an offset measurement in which the packet encounters 4 s of additional delay on the return trip. The calculations in the figure show that this results in a perceived offset of $-2$ s — half the additional delay. Because the entire amount of delay is attributed equally to the outgoing and return trips by the calculations, half of the additional delay in the return trip was incorrectly attributed to the outgoing trip. Thus, the observed offset was in error by the amount incorrectly attributed to the outgoing trip.

### 1.3.2 Information about time quality

In addition, the protocol provides for exchange of information about aspects of the quality of the time provided. There is a presumption that some remote clocks have a better idea of the time than the local system, and the purpose of running an implementation of the protocol is to align the system clock with the remote clocks. In particular, some clocks are labeled "stratum one," indicating that their time is derived from a very good source, usually a radio receiving transmissions from NIST, an atomic clock, or something else that is assumed — outside the scope of NTP, and generally with good reason — to be correct. The protocol provides for comparing multiple sources so that it can provide correct time even while several of the remote clocks fail to provide an indication of time or provide an incorrect indication.

### 1.3.3 Selection of low-delay observations

NTP keeps the observations obtained from the last eight attempts, and only uses the result from the lowest-delay observation. Additional delay beyond the base delay is either on the outgoing trip, i.e., between the time the first and second timestamps are taken, or on the incoming trip, i.e., between the time the third and fourth timestamps are taken, or both. If it is not equally distributed between the two, the offset measurement will be in error. If the additional delay is all outgoing or all incoming, the error will be half the additional delay (see Section 1.3.1 for discussion and Figure 1.2 for an example). Figure 1.3 is a "wedge plot"[1] of offset vs. delay for 1000 offset observations taken immediately in succession. From this we can see that offset errors in fact increase for observations with delay greater than the minimum observed delay. For observations falling along the edge of the wedge, most of the additional delay was either on the outgoing or incoming trip; observations in the middle encountered additional delay on both the outgoing and incoming trips.

While NTP only uses the lowest-delay observation of the past eight attempts, one could not simply greatly increase this value to 1000, for example, as it could cause the suggested mechanism to adjust the local clock to be unstable. In this sense NTP attempts to strike a compromise between using old data and using observations with high delay — neither choice is desirable. Old observations may not reflect the current state of the local clock, and using them may cause unstable behavior. High-delay observations have potentially large errors due to the potential asymmetry in the delays of the outgoing and return trips. These issues are discussed in detail in [Mil92b].

### 1.3.4 Clock selection and combining

The NTP specification suggests methods for selecting which of the remote clocks to use to steer the system clock. The intent is to determine if some of the remote clocks under observation are malfunctioning, i.e., behaving as if their values of time are grossly

[1] The name "wedge plot" and the idea of graphing offset vs. delay is due to David Mills.

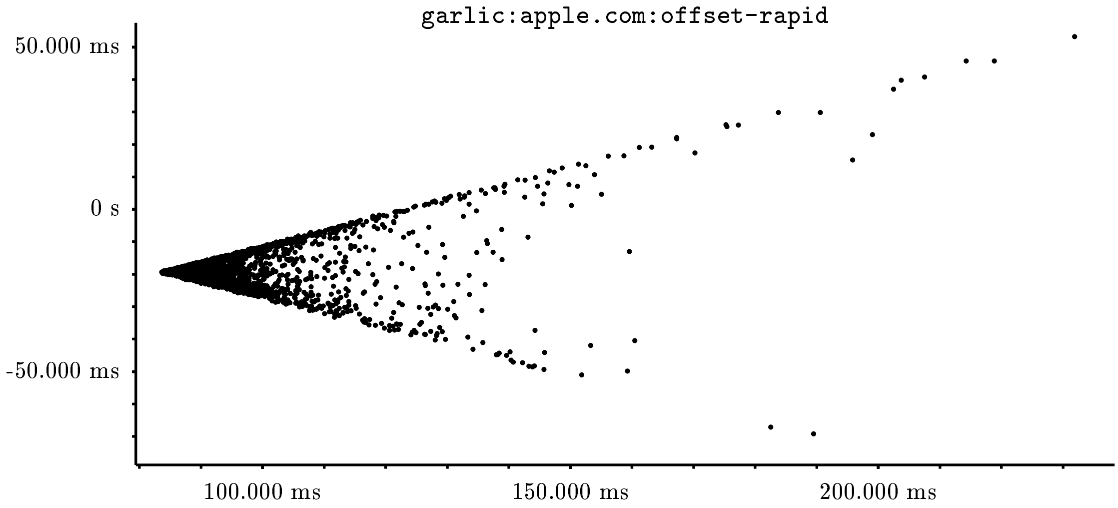



Figure 1.3: Offset vs. delay for 1000 measurements taken in succession of the offset of remote clock apple.com from the experimental computer garlic. Several high-delay points have been omitted to better show the structure of the data in the region of interest.

different from the correct time, and to select clocks that are not malfunctioning. First, a selection process eliminates some clocks because they differ too much from the rest. Then, a combining procedure combines the values of the current lowest-delay offset observations of the remaining clocks to produce a synthetic offset value called the "system offset." This offset value is then used to steer the local clock.

Much of the initial explanation of techniques in this thesis will be in the context of the NTP's system offset, as opposed to the individual offset measurements made to specific remote clocks.

### 1.3.5 Manipulation of the system clock

Another area described in the NTP specification is the way in which the system clock is manipulated given the system offsets. The NTP specification terms this manipulation "local-clock algorithm" or "local-clock mechanism." The basic scheme is a second-order phase-locked loop (PLL) that drives the measured offset (phase error) to zero. The explicit integrator in the loop represents a frequency-error compensation term that is used to correct the frequency of the system's oscillator, as the computer clock almost always exhibits a frequency error. This scheme tends to bring the effective frequency of the computer's clock, i.e., the natural frequency after being corrected, to the correct frequency; when the clock is persistently slow the effective frequency is increased. The actual scheme used by NTP is more complex than the above description; further details may be obtained by reading the NTP specification. These details will be summarized as needed when performance of this scheme is discussed in later sections.

### 1.3.6 Performance of NTP

I later present data showing the performance of NTP under various conditions. Generally, NTP achieves reasonable synchronization performance under widely varying conditions of local oscillator stability, variation in network delay, and behavior of remote clocks. With a local oscillator stable to several parts in $10^8$, the conditions existing in today's Internet, and the remote clocks currently used by the Internet community at large for operational synchronization, I observed the synchronization behavior of NTP. Under some realistic conditions, maximum errors of approximately 20 ms were observed. The synchronization measurement technique presented in this thesis enables an understanding of why better synchronization was not achieved.

## 1.4 Time Surveying

I refer to the synchronization measurement technique and the new synchronization method as Time Surveying because the approach of the two techniques is inspired by and is similar to methods and attitudes of the surveying community. I discuss the nature of surveying, and the ways in which Time Surveying differs. One important and fundamental difference is that Time Surveying is by its nature an ongoing process; it is not sufficient to synchronize a clock once. Other differences relate to the nature of errors present in the measurement tools available to the Time Surveyor. In some ways, many of these differences are not fundamental, but are instead the result of cost-benefit tradeoffs. However, the cost-benefit tradeoffs are different not just because many people might not care about time as much as property, but because Time Surveying is in fact a continuous process.

### 1.4.1 The nature of surveying

One of the tasks of the surveyor is to attempt to determine the true values of various physical quantities, such as locations of objects in the horizontal plane, and their heights. The surveyor makes measurements of these and other quantities; in many cases the number of measurements is greater than the number of values to be determined. Then, the surveyor estimates the values of the desired physical quantities.

#### Measurements and errors

In surveying there is a notion of *direct* and *indirect* measurements ([BHB62], pp. 473–474). The definitions and examples below are paraphrased from those of Breed, Hosmer and Bone. A direct measurement actually measures the quantity of interest. An indirect measurement is the result of a computation of direct measurements. One could determine the distance between two markers with a steel measuring tape; this would be a direct measurement. One could determine distance in the problem above by erecting a third marker near one of the two, making a direct measurement of the distance between the third marker and the closer of the two, directly measuring the angles between the three markers, and solving for the unknown distance; this would be an indirect measurement.

All measurements of physical quantities are affected by errors. In the surveying community, errors are grouped into three broad classes ([BHB62], p. 472):

**systematic errors** "those whose magnitude and sign are definitely related to some condition"

**random errors** "those for which it is equally probable that the sign is plus or minus, and also that small errors are more likely to occur that large ones"

**mistakes** "human errors which must be eliminated by careful work and constant checking"

In this thesis I will use a similar categorization. I will treat improperly functioning computer hardware or incorrect software as either a systematic error or a mistake, depending on the nature of the failure. Breed, Hosmer and Bone consider the random errors relevant to surveying to be *normally distributed*, i.e., according to a Gaussian probability density function. This, however, is not an appropriate model for errors relevant to time synchronization over packet networks. Later, I will examine distributions of observed errors while actually attempting to synchronize clocks over packet networks, and it will be clear that they are not Gaussian. Because the relevant errors are non-Gaussian, I am not able simply to adopt techniques that assume Gaussian errors. Rather, I will use domain-specific knowledge to develop new techniques suitable for the problem. These techniques will declare some observations to be mistakes, and discount the values of such observations.

#### "Adjustment" of observations

In a survey where high accuracy and confidence in the results are desired, the surveyor will often take many more measurements than would be strictly necessary to determine the unknown quantities *if there were no measurement errors* ([BHB62], pp. 6–8). In addition to the measurements, there are constraints on the data. For example, the height difference between two points must be in fact the same when examined via different paths. The surveyor then will have a set of measurements that is overconstrained.

After taking the measurements, the surveyor will cross-check them for mistakes, and either retake or discard measurements which are clearly inconsistent with the rest. Then, the measurements will be generally consistent in that there are no gross inconsistencies due to mistakes, but there will be small inconsistencies due to random or systematic measurement errors.

A surveyor will produce estimates of the true values of the desired quantities. Then, in order to check his work, he will compute estimates of the true values of the quantities he actually measured. For example, given the estimated positions of several points, one can compute an estimate of the distance between two of them or angles among three of them. If the original measurements contained no mistakes, the resulting estimates and the observations will agree closely, differing only due to small measurement errors.

This estimation can be done in several ways; a common method ([BHB62], pp. 71, 479) is "least-squares estimation," in which the estimate is produced by minimizing the possibly weighted sum of the squares of the difference between the actual observed data and the resulting estimates of the values which were directly measured. Alternate methods for obtaining results similar to the least-squares method that are less computationally intensive are presented in surveying texts ([BHB62], p. 517); these often involve incrementally making small "corrections" to the data in order to make it consistent without significantly changing it. These processes are sometimes referred to as "adjustment of observations."

I will refrain from using the terms "adjustment of observations" and "adjusted observations." These terms create confusion as to which values are actually observed and recorded, and which are modifications of observed data. I will adhere to the principle that data are taken, logged, and then never modified in any way. I will do much computation on them, and produce modified versions of what they "should have been," or "would have been," but it should always be clear that I am not changing the original data.

### 1.4.2 Why is Time Surveying different?

The results of my efforts are a greater understanding of the error sources in computer clocks and networks, as well as software to estimate time. Because software may be cheaply duplicated and synchronizing clocks using it does not require paying surveyors to go into the field, it is reasonable to attempt to do a better job of time synchronization with what many would consider to be inadequate tools, since the cost of doing so is less than the cost of "doing the job right," which would likely involve contracting for low-delay network service and buying high-grade oscillators, for example.

One approach to more accurate time synchronization might be to purchase network service with guaranteed delay properties. While this approach might be reasonable if accurate time synchronization was so important as to justify spending large amounts of money for guaranteed network service, it is not suitable for situations requiring better time synchronization performance without large expenditures. There certainly exists the usual engineering trade-off of cost vs. performance in terms of the cost of both network service and local oscillators. I believe that my work allows significantly better performance without substantially increasing costs.

#### Nature of errors

Surveyors generally understand the nature of errors present in many common measurements; texts on surveying ([BH45], [RLT50]) contain discussions of sources of error, means of controlling them, and ways of estimating their magnitude. For example, "Elementary Surveying," by Breed and Hosmer contains a discussion of errors possible in measuring distances with steel tapes ([BH45], pp. 11–16). They discuss how to numerically estimate the errors due to different sources; this discussion is in the context of attacking the problem with an understanding of the required degree of accuracy.

Traditionally the surveyor has strategies for dealing with the three categories of errors. The surveyor attempts to avoid making mistakes in the first place, and then checks for them. If found, they are removed or the measurement retaken. At times, measurements are originally taken twice. Systematic errors are compensated for by understanding their nature, or estimation and correction. The effects of random errors — with presumed independent Gaussian distributions — are mitigated by making more measurements than are strictly needed and using estimation. While it is of course possible for mistakes to go undetected, or for a systematic error to go unrealized and therefore uncorrected, the attitude that errors are either small and Gaussian or are detected and corrected appears common. This seems consistent with a desire to achieve a high degree of accuracy, and to be confident that at the end the desired degree of accuracy has in fact been achieved.

As a Time Surveyor using packet networks, however, the situation is different. The error sources are not quite so easy to categorize, and are less well understood. Errors are often dependent on uncontrollable factors, such as network loading and congestion, that cannot be easily measured directly. For example, variance in network delay is the result of a complex and changing system of routers and links, containing many queues affected by the activities of many other people and programs. Errors in the timekeeping of remote clocks should in theory be rare to nonexistent, but I have found that they occur significantly often. Because I restrict myself to do the best job possible with the available tools, I do not have the luxury of the logical analog of the surveying approach: purchasing better network services and service from a remote time provider in order to meet an accuracy specification.

#### Nature of time vs. location

The nature of time is fundamentally different from that of location — one can return to the same location and make a measurement again, but once time has passed it is gone and data cannot be acquired retroactively. Also, measurement of time is an ongoing process. We do not desire to know where a house or fence *is*, or to place *once* a concrete bound at a given location, but to either know how far off our clock is or to set it right *over a period of time*. This ongoing process will have recurring costs.

Time synchronization over today's packet networks appears to require a much greater number of measurements relative to the number of things which are desired to be known. This is indicative of the relatively large errors present in the measurement processes relative to the desired accuracy.

#### Local oscillator as a measurement tool

One obvious measurement tool is the process of making measurements of the offset to remote clocks over the network. Another necessary tool is the local oscillator itself, the device which is used for timekeeping locally. Rather than viewing the local oscillator as a flawed timekeeping device needing constant corrections, I view it as a critical tool in time synchronization. However, its characteristics are not usually known, and the Time Surveyor must determine them as part of the time synchronization process.

#### Knowledge about inexpensive time surveying tools

Much is known about the stability properties of atomic clocks and the characteristics of very good (and expensive) methods of transferring precise time. Much is known about the error properties of devices used in surveying. However, less information is available on the delay variance characteristics of today's operational packet networks. Also, very little information is available about cheap workstation clocks, other than worst-case specifications. Since it is the nature of worst-case specifications that most clocks often perform better than the worst case, further study seems warranted as to the actual behavior that can be expected.

### 1.4.3 Tasks of the Time Surveyor

The Time Surveyor's tasks include determining how well clocks are synchronized, and why they were not better synchronized. Another task is to synchronize clocks — to ensure that they have the correct value of time, and that they run at the right rate. In addition, there is the problem of determining the difference between the value of a computer's clock and correct time after the fact. One should also consider what the Time Surveyor is simply not capable of doing.

An important, but perhaps overlooked, task is to analyze the performance of existing time synchronization algorithms. With NTP, for example, it is not at all easy to analyze behavior from examining log messages generated by implementations. The Time Surveying techniques presented in this thesis enable much better analysis of the behavior of this protocol, and allow a clearer understanding of the obstacles to achieving better performance with it.

Perhaps the most obvious task is to solve the time synchronization problem in real time — to arrange for a computer's system clock to always have the correct value within some required accuracy. Another task is to solve the time synchronization problem "off-line." By this I mean to later determine the error in the computer's system clock as a function of time. While this may not seem at first useful, if the application is measurement rather than control, this enables the timing of measurements to be more accurately determined for later processing.

In order to accomplish these top-level tasks, the Time Surveyor needs to determine the characteristics of the tools used. These include the local oscillator, the various networks used, and the remote clocks.

#### Impossibility of offset determination in a packet network

In a synchronization system in which a computer may make offset measurements to remote clocks via a network, and the network does not make any guarantees about delivery of packets, it is not possible to achieve precise synchronization of time, but only of frequency. It is not possible for one computer to determine the difference between the value of its system clock and that of another computer solely by making measurements over a packet network. This is because long-term conditions in systematic offset cannot be separated from long-term conditions in asymmetric delays.

Proof:

Consider a set of offset measurements taken over time. Consider two scenarios. Let scenario "A" be what actually happened. Let scenario "B" be the case in which the actual delay on every outgoing trip was 1 ms less than the delay on the corresponding trip of scenario "A," the actual delay on every return trip was 1 ms greater, and the value of the remote clock every time it is read was 1 ms greater than the corresponding value in scenario "A". The values recorded in scenario "B" will be exactly the same as those in scenario "A". The first timestamp will be the same. The second will be taken in "B" 1 ms earlier after it was taken in "A," but by a clock which is 1 ms fast, and hence the value will be the same. The third timestamp will be the same, since the second will be the same. The fourth timestamp will be taken in "A" at the same time as before, since the total delay is the same. Thus, the observed offset and delay values will be the same for the two scenarios in each case. Thus, the first computer may not in fact determine the actual offset via network measurements only. □

In this thesis, I am not overly concerned with exact determination of offset, as it is not possible. I am, however, concerned with frequency stability in local timekeeping.

### 1.4.4 Approaches to time synchronization

Approaches to time synchronization can be divided into two fundamental groups, based on the view taken of the system clock. One views the system clock as an object to be set in time and adjusted in frequency, with the desired result being that it has the correct time and the correct frequency. This is the approach taken by David Mills of the University of Delaware, the developer of the Network Time Protocol. A person attempting to correct a wristwatch using this philosophy would set the watch periodically, and would adjust the frequency-setting trimmer capacitor inside the watch in order to make the periodic corrections smaller or less frequently required.

I take a different approach, inspired by the mindset of surveyors, and view the local oscillator as a measuring instrument, and the system clock as a value to be computed based on the results of the measurements. I realize that the local oscillator is imperfect, and exhibits changing behavior. Instead of attempting to "fix" it, however, I estimate the nature and magnitude of the flaws, and correct for them without modifying the instrument itself. This is analogous to a person who rarely sets his watch, but periodically records its offset error, estimates the frequency error, and estimates the correct time from the indicated time and the estimated offset and frequency errors.

This procedure has in fact used by navigators since approximately 1770 ([Bow84], p. 432). The value of *chronometer error*, the offset error, is recorded at different times, and *chronometer rate*, the frequency error, is calculated. When the chronometer is read, a value of chronometer error for the current time is calculated from the most recent values of chronometer error and chronometer rate. The chronometer is not set or adjusted in frequency, except when it is overhauled and cleaned, perhaps every 3 years.

### 1.4.5 Trusted references

One obvious method for evaluating clock synchronization is to record the difference between the computer's clock and the correct time as a function of time. This can be considered to be an approach of a Time Surveyor who has such a measuring instrument.

If the clock were perfectly synchronized, the values recorded would all be zero. If the clock were imperfectly synchronized, the errors would be easily observable. However, such a method of evaluating synchronization performance has drawbacks, such as cost and operational complexity. I will use such a trusted reference to demonstrate the validity of the analysis and synchronization techniques presented here, but the techniques do not in any way require such a trusted reference.

#### Signatures of various simple errors

It is helpful to consider the results that would be obtained by measurement relative to a trusted reference under various circumstances. If the clock being measured had a constant offset, e.g., 10 ms fast, from correct time, the values recorded would be this constant. If the clock had no initial offset, but had a constant frequency error, the values recorded would be described by a line with zero offset and a slope equal to the frequency error. For example, a clock that is 1 ppm fast would have recorded values falling on the line $e(t) = 10^{-6}(t - t_0)$.

If the clock had no initial offset, no initial frequency error, but a linear rate of change of frequency error, the values recorded would be described by a parabola, with the coefficient of the quadratic term being half the rate of change of frequency error. For example, a clock with a rate of change of frequency of 0.01 ppm per day would have recorded values falling on the line $e(t) = (10^{-8}/(2 \times 86400\text{ s}))(t - t_0)^2$.

Assume for a moment that the clock under study had no initial offset or rate of change of frequency error, but had an intrinsic frequency error of 10 ppm slow. Further assume that this has been previously estimated and the computer is correcting for the error by periodically adding a small amount to the time, so that the average frequency error is zero. This is the correction method used by most implementations of NTP.[2] If we assume that the interval at which the small corrections are made is 1 s, we would find the values recorded for clock error to be a line starting at 0 s and going to -10 $\mu$s at 1 s. At 1 s, there would be a step of 10 $\mu$s, with the recorded value just after the step again being zero. This waveform would then repeat with a period of 1 s.

### Limitations of trusted references

The method of measuring synchronization by comparing the clock under study to an external trusted reference is appealing because it directly captures our intuitive notion, and our definition, of what it means for a clock to be synchronized: that it always have the correct time. However, there are two significant drawbacks to this method.

The first is that it requires an external trusted reference. Use of an external reference usually involves expense and effort to install and maintain. One must obtain a device that provides time. That device must be interfaced to the computer under study. Many computers do not come with facilities to do this easily. In order for the measurements relative to the reference to be meaningful, they must be significantly more accurate than the synchronization being achieved.

Because the comparison method requires a trusted reference, there are many situations where the comparison is simply infeasible for cost reasons, but where an indication of the quality of achieved time synchronization is still desired. Thus, while suitable in the lab, it does not fit into the production environment.

While the comparison method will easily measure the degree to which synchronization was achieved, the other drawback is that it is not easy to answer the next logical question: why was better synchronization *not* achieved? The techniques presented in this thesis provide insight into this issue.

## 1.5 Related work

There has been much work on time synchronization in cases where low-variance time transfer methods are available, and much analysis of very high-grade oscillators. My work differs from this prior body of work in that the time transfer mechanism of packet networks

[2] Errors due to such a mechanism are discussed in [Mil93], p. 6.

has comparatively large errors which may not be easily characterized and that I consider the problem of time synchronization using relatively inexpensive oscillators.

### 1.5.1 Improvements to NTP by David Mills

Dr. David L. Mills of the University of Delaware, the author of the NTP protocol, is also pursuing more accurate clock synchronization ([Mil93], [Mil94]). In his recently published work, he considers error sources when synchronizing clocks from many sources via NTP. Some error sources include serial-line interrupt latency, latency in reading the system clock, and short-term variations in the local clock due to infrequent corrections to account for the estimated frequency error. His work is within the NTP model, and focuses on improving synchronization for computers using NTP. He appears to be focusing on very accurate synchronization for systems with a high-quality source of time, often an attached reference clock.

In contrast, my work focuses on synchronization under poor network performance, particularly in the case where the local oscillator is relatively high-quality as compared to the network observations.

### 1.5.2 The Digital Time Synchronization Service

The Digital Time Synchronization Service (DTSS) is similar to NTP in that it makes offset measurements over a network. It differs from NTP in that it is primarily concerned with providing guaranteed error bounds for the value of the system clock, rather than average-case accuracy. Marzullo and Owicki present ([MO85]) a scheme for determining provable error bounds on values of clocks; this scheme is the foundation of the DTSS. Also, DTSS is designed to make management and configuration of the synchronization process automatic. Because DTSS corrects the value of the local clock, but does not attempt to explicitly correct for frequency errors, the values of time obtained are inferior to NTP in terms of average accuracy. However, DTSS provides a provable error bound.

I see no reason why the techniques used to establish error bounds in DTSS could not be applied to the synchronization scheme described in this thesis.

### 1.5.3 The Smart Clock scheme of NIST

Weiss, Allan, Davis, and Levine describe ([WADL92]) a method for automatically synchronizing clocks to external standards, termed "Smart Clocks." Their paper discusses characterizing a clock by an initial offset, frequency error, and rate of change of frequency error. They discuss remaining errors, such as flicker noise in frequency, and unmodeled frequency error due to the frequency error estimate being incorrect.

Weiss et. al. present an example in which an "inexpensive" quartz oscillator is synchronized via the NIST Automated Computer Time Service (ACTS). This service functions via modems over dial-up lines, and they state that it has a 2 ms measurement noise level. They attempt to maintain a specified accuracy, in this case 1 s, over time intervals between synchronizations as long as one month. Under these conditions it appears that the measurement noise is small compared to both the expected accuracy and unmodeled frequency errors.

The Smart Clock scheme differs fundamentally from this work in that it is based on low-noise measurements to a remote source of time with guaranteed accuracy. Not only does ACTS have a small measurement error of 2 ms, but this error characterization appears known and reliable. Also, the ACTS system always has the correct time, something which is not true of time servers in the Internet world.

## 1.6 Roadmap

In Chapter 2 I present the `rsadj` synchronization measurement technique. I explain the central idea of capturing the right data so that the behavior of the synchronization algorithm being used can be separated from the behavior of the local oscillator, network, and remote clocks. I present a model of the behavior of local oscillators based on physical principles, and discuss how the behavior of a local oscillator can be estimated within the terms of this model. I explain the basic computation of *predicted time* — a hindsight estimate of what the value of the system clock *should have been.* I present examples of the use of the technique, and explain how to interpret the results of analysis.

In Chapter 3 I present the details of the computation of predicted time. While the previous chapter presents fundamental notions of the measurement technique, this chapter further develops the technique to the point where it is useful with realistic data and can cope with the non-ideal behaviors typically observed.

The computation presented in the previous chapter is extended very slightly to become the "overall" computation of predicted time, which is useful for obtaining a broad view of synchronization behavior and the behavior of a remote clock. A version of overall predicted time is computed separately for each remote clock. This chapter also presents the "piecewise" computation of predicted time, also computed for each remote clock. The piecewise computation attempts to accurately characterize the local oscillator, rather than provide a broad overview. It functions in the presence of changes in the local oscillator that are beyond the scope of the basic oscillator model, and identifies regions of time during which the local oscillator may be characterized by the model, and produces a set of estimated oscillator parameters for each region. The piecewise computation can identify and remove from consideration offset observations that are incorrect due to significantly increased delay encountered in the network. It can identify periods in which the remote clock behaved anomalously, and discount observations from such periods. The piecewise computation attempts to strike a balance between errors caused by variance in network delay and by unmodeled changes in the local oscillator.

In addition, this chapter describes the "integrated" computation of predicted time. This third form of computation uses data from multiple remote clocks and attempts to characterize the very recent past, and, by extrapolation, the immediate future. It is useful for comparing the behavior of multiple remote clocks; given several well-behaved clocks and one which is erroneous, integrated predicted time based on the correct clocks can be used to examine the behavior of the erroneous clock. Systematic differences in offsets among remote clocks are estimated.

Chapter 4 presents observations of the components used in synchronization and of the synchronization performance of NTP. I describe the equipment used in the thesis research. I present data using the `rsadj` technique that indicates the stability or lack thereof of several local oscillators, and explain how the measurement technique allows me to draw such conclusions. I present data that indicates poor system clock behavior — the failure to properly accumulate ticks of the local oscillator. I present data about network delay variance, and show the effects of such variations. I present data about several remote clocks, and show that some remote clocks are well-behaved in that they appear to have the correct time. I show that others suffer from various problems, and identify the nature of the errors. I present data about several well-behaved remote clocks in order to show typical "good" behavior. I present data showing the behavior of a low-frequency radio timecode receiver, and that of a GPS timing receiver.

I present data showing the behavior of the Network Time Protocol. I show that NTP is adversely affected by differences in perceived offsets among remote clocks, and that this is in fact a major impediment to accurate synchronization in the operational environment of today's Internet. I show NTP's response to a relatively small number of incorrect offset observations due to high delays. I suggest improvements to NTP's algorithms to avoid the adverse effects of using high-delay observations, to ameliorate the effects of systematic offsets among reference clocks, and to better take advantage of the stability of the local oscillator.

I explain how I used the trusted reference technique to verify the `rsadj` measurement technique, and present corresponding data from a GPS timing receiver and the `rsadj` technique. I show that these data are very consistent.

In Chapter 5 I present the integrated timekeeping scheme. Integrated timekeeping is based on the integrated computation of predicted time presented in Chapter 3. The central idea is to compute predicted time periodically. Given periodic updates to the estimated parameters of local oscillator, the system clock is commanded to take on values computed from the estimated local oscillator parameters and the current value of the local oscillator. I describe subtle problems that arise when computing integrated time for synchronization purposes, and my solutions to these problems.

I present data showing the results of simulating the behavior of integrated timekeeping. By simulating the behavior rather than implementing it, a direct comparison of the behavior of integrated timekeeping and NTP is possible for the *same input data*. I also present data showing the results of an implementation of integrated timekeeping. The performance of integrated timekeeping is analyzed with the `rsadj` technique and by using a GPS receiver as a trusted reference. I show that integrated timekeeping performs approximately an order of magnitude better than NTP given an oscillator stable to a few parts in $10^8$ and delay and systematic offset variations found in today's Internet.

In Chapter 6, I discuss the design of a hardware module for timekeeping constructed for this research. I argue that only a few simple features are necessary for precision timekeeping — essentially a stable oscillator and a counter to accumulate ticks of that oscillator. I argue that the cost of such features other than a stable oscillator are very small, and that such features should be included in computers.

In Chapter 7, I summarize the thesis. I make suggestions for future work. I suggest ways to improve the operational time synchronization of today's Internet.

# Chapter 2
# The `rsadj` Synchronization Measurement Technique

In this chapter I present a new method, the `rsadj` synchronization measurement technique, for observing the behavior of some clock synchronization schemes, particularly the Network Time Protocol (NTP, described in [Mil92b]). Any scheme which makes offset measurements over the network and adjusts the system clock may be observed. The `rsadj` technique allows one to determine how well the clock is synchronized without an external trusted time reference, and allows one to determine in many cases why only the observed degree of synchronization was achieved. A scheme to synchronize clocks based on this method, termed integrated timekeeping, is discussed in Chapter 5.

The name `rsadj` is abbreviation for the words *running sum [of] adjustments*, which describe a central idea of the scheme. The essence of the `rsadj` synchronization measurement technique is to collect the values of offset observations as they would have been in the absence of a synchronization algorithm, as well as information about the adjustments to the system clock made by the synchronization algorithm. This allows the measurement technique access to the "raw" data, unobscured by the behavior of the synchronization algorithm running at the time. It also allows observation of the adjustments made to the system clock; such adjustments are precisely the behavior of the synchronization algorithm. The scheme completely decouples the behavior of the system clock (and thus the synchronization algorithm being run) from the behavior of the aggregation of the local oscillator, network, and remote clocks. Later, I will discuss ways that the behaviors of the local oscillator, the network, and remote clocks can be further decoupled.

The `rsadj` technique is inspired by the attitude in the high-accuracy surveying community that the best measurements possible should be taken, and then the values of interest should be obtained by estimation. The fundamental idea of not adjusting the system clock, but instead obtaining data relative to the never-adjusted local oscillator directly reflects this attitude; it starkly differs from the approach taken by NTP.

After giving an informal overview of the `rsadj` technique, I provide definitions of terms used in the presentation of the technique. I discuss the difficulties inherent in considering various values as a function of time, when time, in fact, is the unknown quantity, and my solution to these difficulties.

I explain how to compute *predicted time*, an estimate of the values the system clock should have had. I discuss the assumptions that must be made in order to estimate parameters of the local oscillator from offset observation information. I explain that rather than computing predicted time itself, I compute the values `rsadj` should have had, specifying the values of the system clock with respect to those of the local oscillator. I present a model of the behavior of local oscillators based on physical principles. I describe the method used to produce estimates of the local oscillator parameters.

## 2.1 The Basic `rsadj` Technique

This section describes the motivation and begins to explain the ideas behind the **`rsadj`** technique. It is not entirely rigorous, and should not be viewed as a definitive explanation of the technique. Rather, it is intended to convey the basic idea to make the detailed and careful presentation of later sections easier to comprehend.

For the rest of this chapter, I will implicitly use NTP's system offset as the input data.

This document includes many graphs, which appear in figures. While the figures have descriptive captions, each graph also has a title. Most titles consist of three components, separated by colons. The first component gives the name of the computer on which the measurements are taken. The second gives the name of the remote clock to which the data refers, or is "NTP" if the graph is of or is derived from NTP's system offset. In some cases the second component will be omitted for graphs derived from NTP's system offset when such a label would be confusing rather than helpful. The third component identifies the quantity being shown. For example, "**`garlic:apple.com:aoffset`**" indicates a graph of offsets of apple.com observed by garlic.

### 2.1.1 Motivation

Implementations of NTP can either be configured or easily modified to log observed offsets relative to remote clocks and estimates of frequency error. If one assumes that the remote clock is correct, the measured offsets are similar to the comparisons to a trusted clock, except that they are contaminated by noise due to varying delays in the network. I observed these offsets and the frequency error estimate over time.

From my previous experience using quartz crystal oscillators, I did not believe that the changes in NTP's frequency estimate over time scales of several hours accurately reflected changes in the local oscillator's frequency. Figure 2.1 shows an example of such data. I postulated that most of the short-term variance in the frequency estimate was due to the effects of network measurement noise on the NTP synchronization algorithm, rather than to actual changes in the frequency of the local oscillator. However, the offset measurements had been used to modify the local clock's effective phase and frequency, and the actual behavior of the local oscillator was obscured by these changes.

### 2.1.2 Keeping track of adjustments to the system clock — `rsadj`

I modified an NTP daemon (a program implementing NTP) to keep track of all adjustments made to the system clock, so that all offset measurements could be transformed to what they would have been in the absence of a synchronization algorithm. The modification consisted of adding a new variable, called **`rsadj`**, short for *running sum [of] adjustments.* At initialization, **`rsadj`** is set to zero. Whenever a change is made to the phase of the system clock for any reason, the same change is made to **`rsadj`**. The modified daemon logs this variable often, essentially whenever it logs any other data. For example, if NTP adds 1 ms to the system clock (in order to correct a perceived error that the clock is slow), 1 ms is added to **`rsadj`**. If NTP's estimate of the frequency error of the local oscillator is 7 parts per million (ppm) slow, i.e. an error of $7 \times 10^{-6}$ slow, then once a second NTP will add 7 $\mu$s to the system clock, and also to **`rsadj`**. Thus, at any time, the value of the system clock

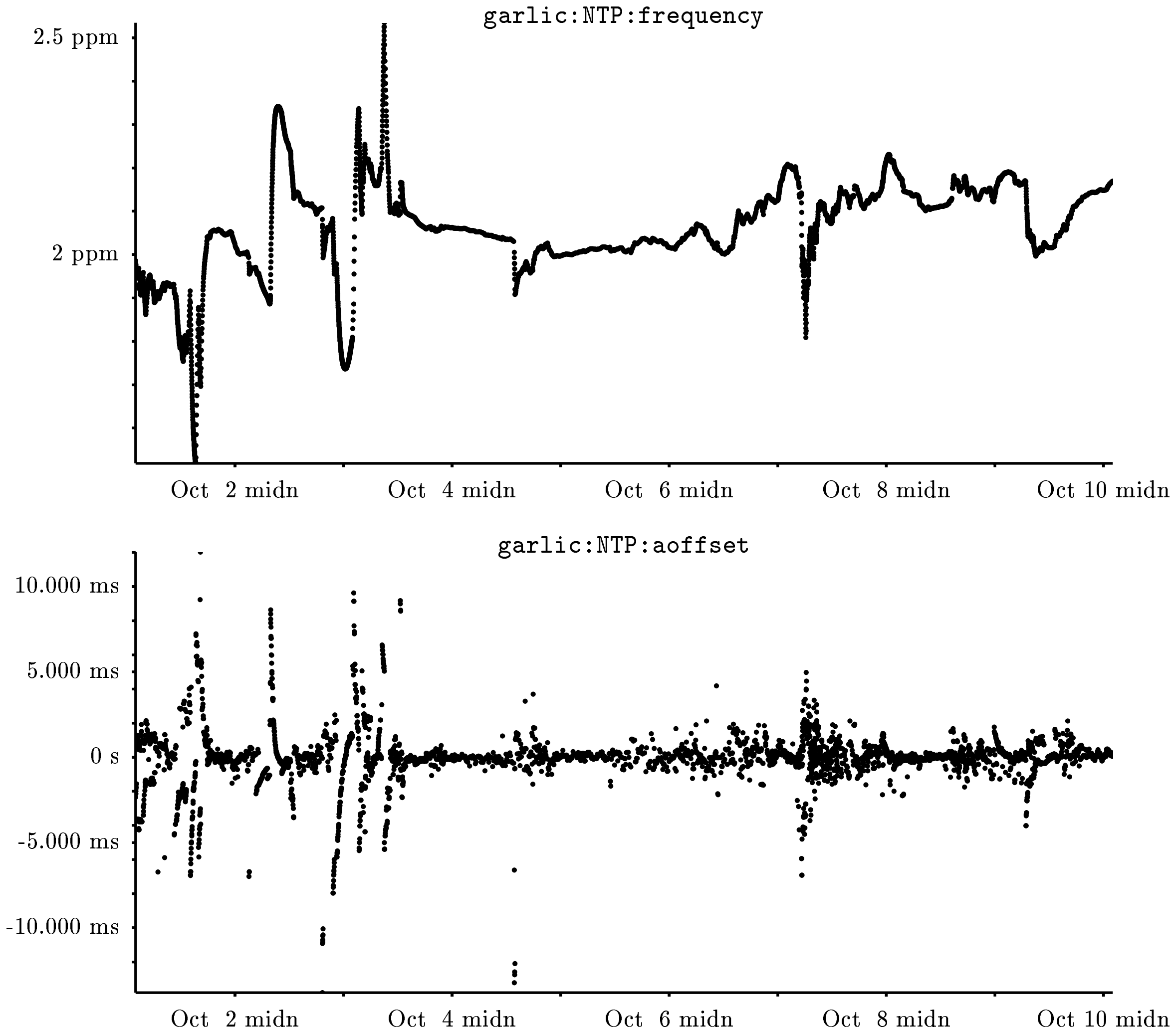



Figure 2.1: Frequency error estimate and offset measurements generated by NTP.

minus `rsadj` is the value the clock would have had if there had been no synchronization activity.

The value of `rsadj` is fundamental in the sense that it describes the behavior of the synchronization algorithm with respect to the local oscillator. The use of the value of `rsadj` to compute the values offset observations *would have had* in the absence of a synchronization algorithm is not fundamental; if in a particular computer values of the local oscillator could be read rather than only values of the system clock, offsets relative to the local oscillator could be recorded directly.

Examination of Figure 2.2, a graph of `rsadj` as a function of time, indicates an approximately linear relationship. Casual inspection of the figure indicates a rate of increase of `rsadj` of roughly 2 s in 5 days, or about 5 ppm. Thus, on average, roughly 5 $\mu$s was added to the system clock every second. Examining a graph of the offsets measured by NTP over the same time period shows that these offsets are small, on the order of 10 ms or less,

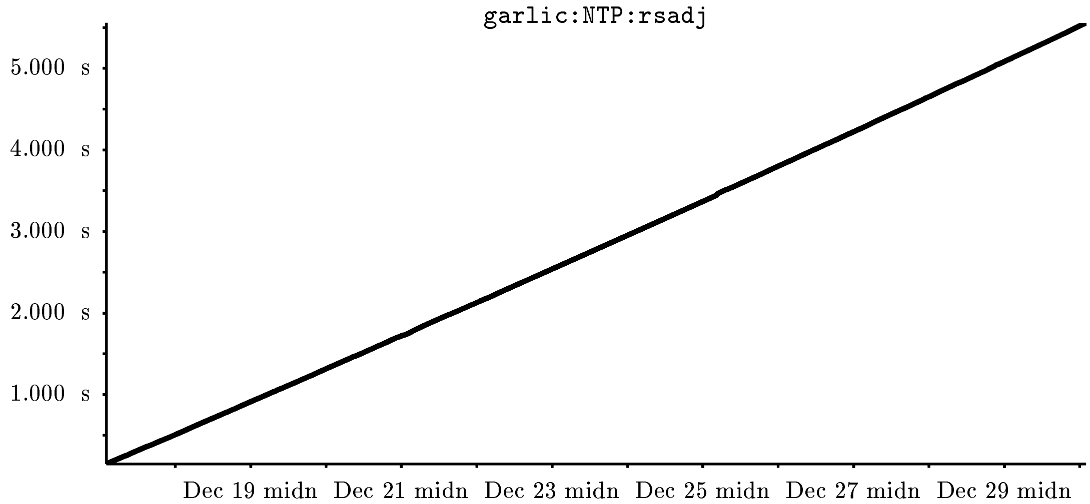



Figure 2.2: `rsadj` vs. time for garlic. While the graph appears to be a line, it is in fact a collection of points.

indicating that the computer's clock remained synchronized to at least approximately this level. I conclude from this data that the clock of this computer is roughly 5 ppm slow, as the clock kept reasonable time when a long-term correction of roughly +5 ppm was applied by the synchronization algorithm.

### 2.1.3 *Uncorrected* observations

NTP periodically makes offset measurements to remote clocks. The offset measurements made reflect the difference between the remote clock's time and the local computer's time (value of the system clock), plus errors due to the network. However, the time of the local computer is constantly being modified by the NTP local-clock algorithms to keep it correct, obscuring the behavior of the local oscillator. One can calculate the value of *uncorrected* offset measurements, denoted `uoffset`, which are the offsets which would have been observed had the system clock not been subject to any adjustments. This calculation is simple given the actual offset observed and the running sum of adjustments, i.e., `rsadj`, to the system clock. For example, if an offset of 2 ms is observed, meaning the remote clock appears to be 2 ms fast relative to the system clock, and 3.4 s has been added to the system clock, the uncorrected offset is 3.402 s, since had no adjustments been made, the system clock would have had a time 3.4 s less than what it did.

Figure 2.3 shows values of uncorrected offsets as a function of time. Rather than offsets to a specific remote clock, the offset used is NTP's system offset, which is a combination of the offsets to a number of remote clocks. This figure is very similar to the previous graph of `rsadj` as a function of time. This is not surprising; the computer was being synchronized using the standard NTP algorithms, so the actual offsets recorded are all very small compared to the scale of the graph..

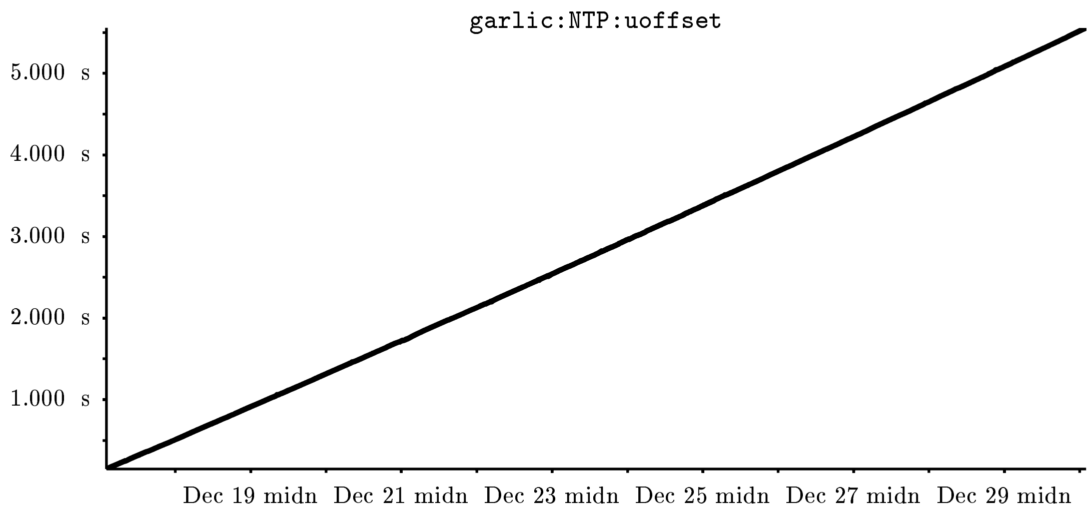



Figure 2.3: `uoffset` vs. time for garlic. As in the previous figure, the graph consists of many points that appear as a line.

### 2.1.4 Modeling of the local oscillator

I consider the local oscillator to have an initial phase error, a frequency error, and an aging rate (rate of change of frequency). Other defects of real oscillators, such as changes in characteristics due to temperature, shock, power transients, or anything else are not included in the model. In Section 3.4, I will present a mechanism for dealing with the possibility of such effects.

The initial phase error is the difference between the indicated time and the correct time at the beginning of the period of interest. The frequency error is the difference between the oscillator's actual rate and the correct rate (at the beginning of the period). The aging rate is the time derivative of frequency error, and is sometimes in this work simply assumed to be zero and not estimated; this issue is discussed later in section 3.6.

Given a number of uncorrected offset observations, one may attempt to estimate the parameters of the system clock by making the assumptions that the uncorrected offsets are in general correct and that the clock can be characterized over the time period of interest by the given parameters. Essentially, one attempts to choose phase, frequency error and aging estimates such that the difference between the function specified by the estimates and the uncorrected offsets is as small as possible. Consider a clock with an initial phase error of 1 s slow and a frequency error of 1 ppm slow and a set of noise-free measurements made to a remote clock having exactly the right time. If we let $p = 1$ s, and $f = 1$ ppm, and subtract $p + ft$ from the values of `uoffset`, the result will be zero in each case. All other choices will produce higher residual offsets.

Recall that Figure 2.3 exhibits a pronounced linear trend, attributed to the frequency error of the oscillator. I performed a least-squares estimation on this data, calculating phase and frequency error estimates which resulted in the smallest sum of squares of the residuals of the data minus the estimate. More precisely, given pairs of times and uncorrected offsets

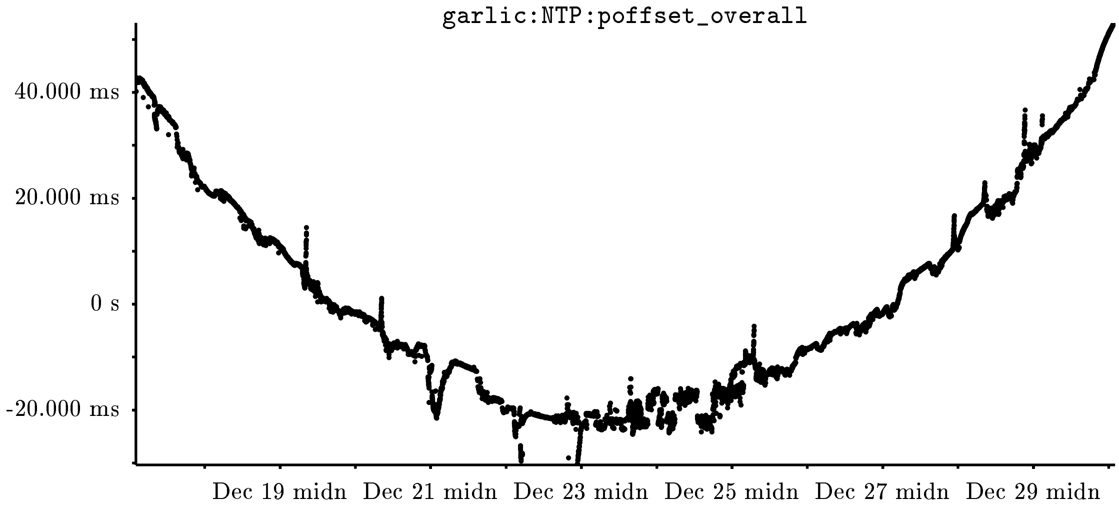



Figure 2.4: `uoffset` – `predrsadj` vs. time for garlic (linear model)

$(t_i, o_i)$, the phase and frequency error estimates are $\hat{p}$ and $\hat{f}$, such that

$$\sum_i (o_i - (\hat{f}t_i + \hat{p}))^2$$

is minimized. In this case, I call the quantity $\hat{f}t_i + \hat{p}$ the *predicted* running sum of adjustments, or `predrsadj`. The results were a phase of 0.107 s and a frequency error of 4.81 ppm. The rms error — the square root of the sum of the squares of the residuals (`poffset`) — was 19706 $\mu$s. Figure 2.4 shows the resulting residuals, i.e., the original values of `uoffset` minus the values predicted by the estimation (`predrsadj`). This figure still shows a distinct trend. Rather than an obvious line, the data appear parabolic. Note, however, that the scale of the dependent variable is drastically reduced, and features that were obscured in the earlier graphs are now apparent.

I performed another least-squares estimation on same data, this time calculating phase, frequency error, and aging estimates. In this case given pairs of times and uncorrected offsets $(t_i, o_i)$, the phase, frequency error, and aging estimates are $\hat{p}$, $\hat{f}$, and $\hat{a}$, such that

$$\sum_i (o_i - (\frac{\hat{a}}{2}t_i^2 + \hat{f}t_i + \hat{p}))^2$$

is minimized. The results were a phase of 0.150 s, a frequency error of 4.57 ppm, and an aging rate of 0.037 ppm/day. The rms error was 2117 $\mu$s, much smaller than in the linear case. Figure 2.5 shows the residuals from the second estimation. In this figure, no particular long-term trend is clearly present.

### 2.1.5 Use of hindsight estimates as an analysis tool

If I knew exactly the phase, frequency error, and aging rate of the local oscillator, I could constantly adjust the system clock to have the correct time by compensating for the known errors. This could be done by making adjustments to the system clock so that `rsadj`

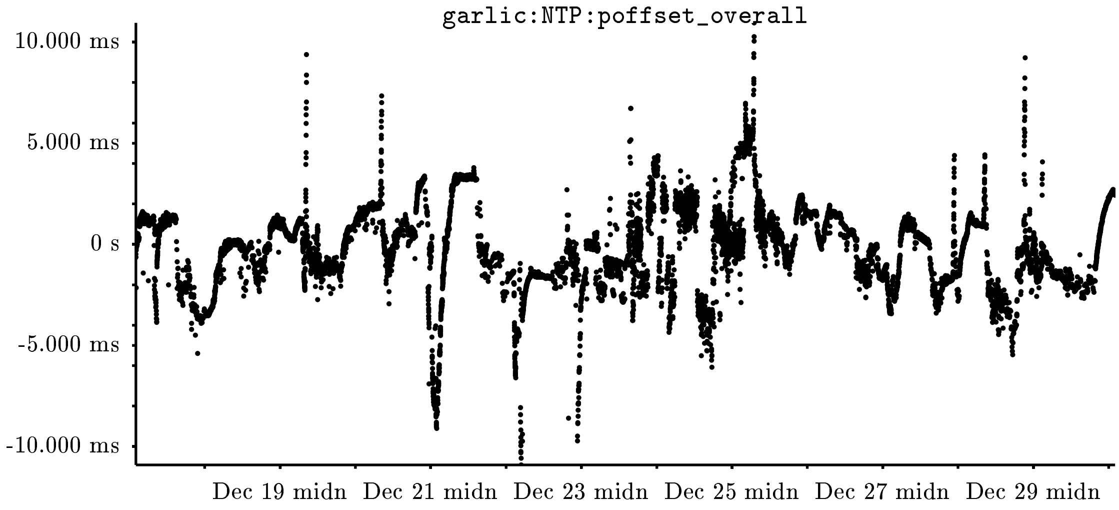



Figure 2.5: `uoffset` − `predrsadj` vs. time for garlic (model with aging)

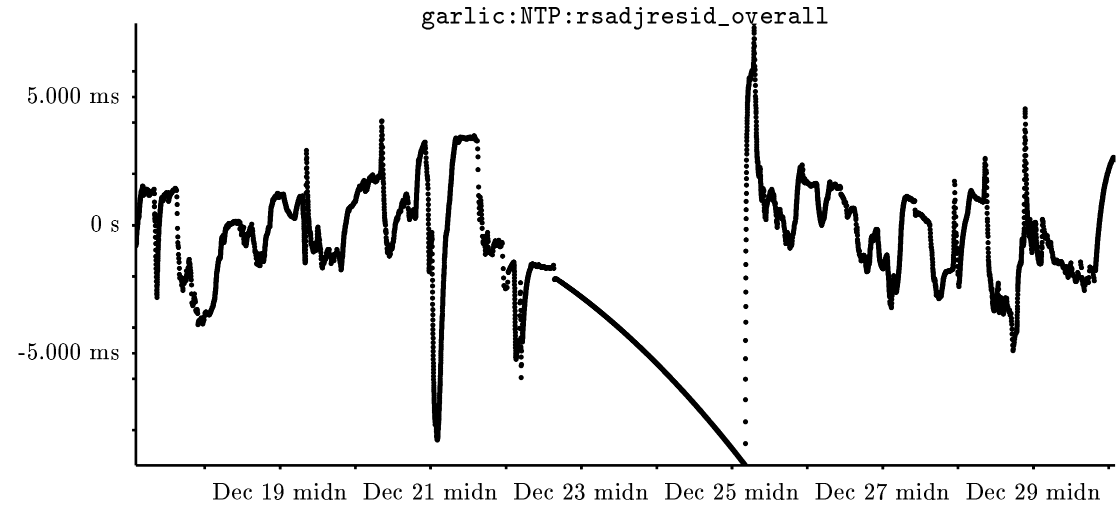



Figure 2.6: `rsadjresid` (in seconds) vs. time for garlic (model with aging)

is, for all $t$, equal to $\frac{\hat{a}}{2}t_i^2 + \hat{f}t + \hat{p}$, or `predrsadj`. Rather than attempting to do this for the future, I consider here what *would have* happened had the clock been controlled in this manner. I of course do not have access to actual offset measurements taken with the clock adjusted this way, since it was in fact not adjusted this way, but can easily calculate what the measurements would have been. I have already performed this calculation — the residuals from the estimation of the clock parameters are precisely the offsets that would have been observed had the clock been steered in accordance with the estimates. This will be shown more rigorously later, but this result is not surprising — the quantities minimized during the estimation were the differences between the uncorrected offsets and `predrsadj`.

I can also calculate the difference between the adjustments that were made and those that would have been made; Figure 2.6 shows that data. This graph shows the difference between the value to which the system clock would have been set and the value it actually did have. For most of the time represented by this graph, the system clock was under the control of NTP. From late on December 22 until just after midnight December 25, however, the system clock was commanded to instead follow the results of a prior estimation. While the system clock steadily drifted away from the hindsight-preferred value, its short-term jitter was very low. Note however, that both of the above statements really contain an implicit *with respect to the local oscillator* qualification.

From examining plots of the residuals of uncorrected offsets with respect to the estimated parameters, and also plots of the difference between actual and predicted adjustments, much insight can be gained into synchronization performance. The effects of the synchronization algorithm that actually controlled the system clock during the period being examined (Figure 2.6) are decoupled from the raw observations (Figure 2.5). This enables the observer to draw inferences regarding remote clock behavior, network behavior, local oscillator behavior, and the response of existing synchronization algorithms to these behaviors. Chapter 4 explains and documents many such observations.

## 2.2 Definitions for the `rsadj` technique

In the preceding section I have used the terms such as "uncorrected offset" without rigorously defining them, in order to present the `rsadj` method informally with the hope of making the formal presentation easier to comprehend. Here, I present careful definitions for the quantities under consideration.

### 2.2.1 True time and other kinds of time

*True* time, denoted `ttime`, is the *correct* value of time at any point. This quantity is not actually available to the measurement scheme. It is, however, used as the independent variable — most other quantities are considered functions of true time. I will write, for example, `uoffset`, to refer to the set of ordered pairs (`ttime`$(t)$, `uoffset`$(t)$), will write `uoffset`$(t)$ to refer to the value at a particular time, and will write $(t_i, u_i)$ when I wish to treat the mapping as a set.

A small problem is that quantities are a function of `ttime`, but that `ttime` is not available. Thus, we use *actual* time instead, and argue later in section 2.3 that this difference is inconsequential.

*Actual* time, denoted `atime`, is the value of the system clock at any point in true time. This is the value that would be returned by a system call requesting the current time, for example. When referring to time on both remote and local systems, actual times of the local and remote system will be written $\texttt{atime}_{\texttt{lcl}}$ and $\texttt{atime}_{\texttt{rem}}$.

*Uncorrected* time, denoted `utime`, is the value that the system clock would have at any point in true time if there had been absolutely no corrections made to the system clock.

*Predicted* time, denoted `ptime`, is the value that the system clock would have at any point in true time if corrections had been made to the system clock in a particular specified manner. A value of `ptime` is a mapping from `ttime` to `ptime`. There is no one value, but a value given a particular scheme for specifying corrections; `ptime` will only be discussed in the context of particular schemes, of which I will present several.

### 2.2.2 Different varieties of rsadj

The varieties of `rsadj` will be referred to as *time trajectories*. Each can been viewed as a mapping from `utime` to `atime` or `ptime`, and as such each kind of `rsadj` is a function of time.

*Running sum of adjustments*, denoted `rsadj`, is the difference between actual time and uncorrected time:

$$\texttt{rsadj} = \texttt{atime} - \texttt{utime}$$

In other words, at any point in time, it is the sum of all changes made to the system clock up to that time.

*Predicted running sum of adjustments*, denoted `predrsadj`, is the difference between predicted time and uncorrected time:

$$\texttt{predrsadj} = \texttt{ptime} - \texttt{utime}$$

*Residual of* `rsadj`, denoted `rsadjresid`, is the difference between `rsadj` and `predrsadj`:

$$\texttt{rsadjresid} = \texttt{rsadj} - \texttt{predrsadj}$$

Expanding the definitions of `rsadj` and `predrsadj`, one can see that `rsadjresid` is also the relationship between *actual* and *predicted* time:

$$\texttt{rsadjresid} = \texttt{atime} - \texttt{ptime}$$

### 2.2.3 Different varieties of offsets

A clear understanding of the meaning of "offset" is central to understanding NTP and particularly the `rsadj` technique. As discussed earlier, NTP makes measurements of the difference between the time of a remote computer and the local computer's time. These measurements are noisy, and the principal sources of measurement noise is variance in the round-trip network delay, and asymmetry in the outgoing and return trip delays. NTP makes offset measurements to various remote clocks on a regular basis (an attempt every 64 to 1024 seconds).

Since in general the *uncorrected* and *predicted* time of the remote computer are not available, I must in these definitions and in our work use the remote computer's *actual* time.

*Actual* offsets, denoted `aoffset`, are the *observed* difference between a remote computer's *actual* time and and the local computer's *actual* time, and are thus contaminated by measurement noise:

$$\texttt{aoffset} = \texttt{atime}_{\texttt{rem}} - \texttt{atime}_{\texttt{lcl}} + \texttt{noise}$$

This is precisely what is measured by NTP and used for the standard NTP algorithms.

*Uncorrected* offsets, denoted `uoffset`, are the observed difference between a remote computer's *actual* time and the local computer's uncorrected time:

$$\texttt{uoffset} = \texttt{atime}_{\texttt{rem}} - \texttt{utime}_{\texttt{lcl}} + \texttt{noise}$$

In other words, a `uoffset` is the offset that would have observed had no adjustments been made to the system clock, that is, had the system clock read *uncorrected* time rather than *actual* time).

*Predicted* offsets, denoted `poffset`, are the observed difference between a remote computer's *actual* time and the local computer's predicted time:

$$\mathtt{poffset} = \mathtt{atime}_{\mathtt{rem}} - \mathtt{ptime}_{\mathtt{lcl}} + \mathtt{noise}$$

In other words, a `poffset` is the offset that would have been obtained had adjustments been made to the system clock in a particular specified manner, that is, had the system clock read *predicted* time rather than *actual* time).

## 2.3 What does time as a function of time mean?

During much of this work I have referred and will refer to various quantities, such as offset measurements to remote computers, as a function of time. While this may at first seem straightforward, it may also seem peculiar and indefensible on closer examination, since the independent variable time is precisely the object of study and we do not presume to know it exactly. This section provides an explanation of why the practices I adopt in this regard are sound.

In this work I shall always use `atime` as the independent variable instead of `ttime`. The difference between actual time and true time is almost always very small — because the computer under study is either running the NTP algorithm or the synchronization algorithm presented in Chapter 5, the system's *actual* time is always reasonably close to *true* time.

As a hypothetical but realistic example, consider two offset measurements made one day apart on a machine where the oscillator was in error by approximately 50 ppm slow. Assume that the machine was reasonably well synchronized over this period. Assume without loss of generality that the value of `rsadj` was 0 s at the time of the first measurement, and let us say that the offset measurement was $-1$ ms, i.e., the remote clock appeared to be 1 ms slow. That is, say that `aoffset` and `uoffset` were both $-1$ ms. Let us say that the second offset measurement is taken when the system clock reads exactly 86400 s greater than when the first was taken, and that the result is 2 ms. Let us say that the value of `rsadj` is 4.32 s at this time. In this case, `aoffset` is 2 ms, and `uoffset` is 4.322 s. If I accept `actual` time as the independent variable, I obtain a frequency estimate of $(4.322 - (-0.001))$ s/86400 s $= 50.0347$ ppm. If I instead use the uncorrected times of the measurement, i.e., 0 s for the first measurement, and $86400 - 4.32$ s for the second, I obtain a frequency estimate of 50.0372 ppm. In this case, the difference in the estimates is very small — far smaller than the likely errors in the estimates themselves. For example, had the second offset measurement been 1 ms rather than 2 ms, the frequency estimate would have been $(4.321 - (-0.001))$ s/86400 s $= 50.0231$ ppm.

However, the second frequency estimate above is in fact wrong. A frequency error estimate is a rate of change of the difference between a clock's value and true time. Implicit in this definition is that the rate of change is with respect to true time. Thus, I should be using true time as the independent variable, which, of course, is not available. Thus, I use the best estimate available, which in our case is actual time. Note that actual time is almost always much closer to true time than uncorrected time. One may easily observe that the *actual* offsets recorded are all less than 1 s in magnitude. Thus, I conclude that `atime` is always close to `ttime`. The above example demonstrates that even an error of 4 s — far greater than those observed — is not serious.

While the experimental software records observed offsets and values of `rsadj` to the microsecond, the times at which these offset measurements were made are recorded with a precision 1 s. Even with frequency errors of 100 ppm, however, the difference in predicted `rsadj` is only 100 $\mu$s over this time. Offset measurements are almost always far noisier than this. The time difference between the beginning and end of a particular offset measurement can easily be 100 ms or more.

## 2.4 Recorded data and derived data

Implementing the `rsadj` technique requires keeping sufficient data so that needed quantities have either been recorded or can be calculated. The quantities needed are `aoffset`, `uoffset`, and `rsadj`. In my work, `aoffset` and `rsadj` are recorded as a function of actual time. One may derive an expression for `uoffset` in terms of `aoffset` and `rsadj`:

$$\texttt{uoffset} = \texttt{atime}_{\texttt{rem}} - \texttt{utime}_{\texttt{lcl}} + \texttt{noise}$$

$$\texttt{utime}_{\texttt{lcl}} = \texttt{atime}_{\texttt{lcl}} - \texttt{rsadj}$$

$$\texttt{uoffset} = \texttt{atime}_{\texttt{rem}} - (\texttt{atime}_{\texttt{lcl}} - \texttt{rsadj}) + \texttt{noise}$$

$$\texttt{uoffset} = (\texttt{atime}_{\texttt{rem}} - \texttt{atime}_{\texttt{lcl}} + \texttt{noise}) + \texttt{rsadj}$$

$$\texttt{uoffset} = \texttt{aoffset} + \texttt{rsadj}$$

It may seem counterintuitive that `rsadj` is added rather than subtracted, but the definition of offset used by NTP and which I have adopted is the time of the remote clock minus the local clock time. For example, if an offset measurement of 4 ms is made, and `rsadj` has the value 2 s, this means that 2 s has been added to the system clock, and that it thus would have been 2 s behind its current value, and thus the offset measurement would have been 2004 ms.

## 2.5 Basic computation of *predicted* time

When defining *predicted* time, I referred to the time the system would have had if "corrections had been made to the system clock in a particular specified manner." The definition is intentionally unspecific because I will present several ways to determine the desired corrections, depending on the intent. There are several reasons to compute *predicted* time. One is to determine how well the synchronization algorithm being run works, and another is to synchronize the clock.

The basic concepts in the computation of *predicted* time are to assume that the local clock can be characterized by the phase, frequency, and aging model and to attempt to make `ptime` $-$ `ttime` small for all times. For analysis, one might desire that this condition be equally true for all times in the past, and not particularly care about the immediate future. For synchronization, one would likely desire that this condition be true for a specific future interval, perhaps the next four hours or so. This section presents the most simple method; later chapters present specific methods for analysis and synchronization. They also address the situation when the assumption that the local clock can be simply characterized does not hold.

I would like to calculate ptime such that ptime − ttime is small. This is problematic for two reasons. One is that specifying ptime as a function of ttime is not reasonable when ttime is not physically accessible. The other is that attempting to minimize ptime − ttime is not reasonable for the same reason.

### 2.5.1 Calculate predrsadj rather than ptime

As a solution to the first difficulty, I calculate predrsadj = ptime − utime rather than ptime itself. Once predrsadj is calculated, I may calculate

$$\mathtt{poffset} = \mathtt{uoffset} - \mathtt{predrsadj}$$

and

$$\mathtt{rsadjresid} = \mathtt{rsadj} - \mathtt{predrsadj}$$

Later chapters will show that there is no need to actually specify ptime directly; the quantities used are predrsadj, poffset and rsadjresid. I briefly argue that specifying predrsadj is sufficient for both analysis and performing synchronization.

When analyzing synchronization performance, I wish to find out what would have happened had the clock followed the "correct" time trajectory, rather than the one it did. Since predrsadj is the difference between ptime and utime, and since the values of utime and atime are known at each measurement, I can calculate ptime as of that measurement. What is still unknown is the relationship between ptime and ttime. I argued in section 1.4.3 that this cannot be exactly determined by a time synchronization scheme that may only make offset observations over the network.

When using this technique to synchronize the clock, the desired output is exactly predrsadj, the mapping between utime and ptime. In this case, the timekeeping daemon would be instructed to cause rsadj to equal predrsadj, so that atime equals ptime.

### 2.5.2 Minimize poffset rather than ptime − ttime

Above, I stated that I want to make ptime − ttime small, but this cannot be done directly because I do not have access to ttime. In considering solutions to this problem, let us step back and consider the data that is available, the nature of the problem of time synchronization over packet networks, and the assumptions that are normally made.

I first assume that the remote clock to which offset measurements are being made has the correct time, i.e., that $\mathtt{atime}_{\mathtt{rem}} = \mathtt{ttime}$. While it can be argued that this may not be correct, such an assumption is at the very foundation of NTP and similar schemes; the entire point of the scheme is to synchronize one's clock to the correct time by making offset measurements over the network to remote clocks presumed to have the correct time. Later, I will relax this assumption.

I have assumed that the remote clock is correct, and that I would like ptime to be close to ttime. Let us consider again the definition of poffset:

$$\mathtt{poffset} = \mathtt{atime}_{\mathtt{rem}} - \mathtt{ptime} + \mathtt{noise}$$

Incorporating the assumption that $\texttt{atime}_{\texttt{rem}} = \texttt{ttime}$, we obtain

$$\texttt{poffset} = \texttt{ttime} - \texttt{ptime} + \texttt{noise}$$

Since I would like $\texttt{ptime} - \texttt{ttime} = 0$, I would like to set ptime such that

$$\texttt{poffset} = \texttt{noise}$$

This is clearly not possible, since I do not have access to the value of noise. Thus, I consider letting $\texttt{ttime} - \texttt{ptime} + \texttt{noise}$, rather than $\texttt{ttime} - \texttt{ptime}$, be zero. In this case, the desired outcome is $\texttt{poffset} = 0$. This is achievable — I simply set predrsadj equal to uoffset. Now poffset is 0 for all $t$, because by definition $\texttt{poffset} = \texttt{uoffset} - \texttt{predrsadj}$.

In effect, this says that the local clock should have been steered such that each offset measurement would have been zero. A synchronization scheme based on the above method would simply add the result of each offset measurement to the system clock. However, such a scheme is unreasonable because it essentially assumes that every offset measurement is correct and that the local oscillator exhibits arbitrary perturbations. The first assumption is simply not correct, since offset measurements contain noise. Recall that Figure 1.3 showed roughly 1000 offset measurements taken in rapid succession; results ranged from $-70$ ms to $+50$ ms. Particularly troubling is the notion of completely arbitrary local oscillator behavior; this assumption is grossly inconsistent with everyday experience with timekeeping devices based on the same sort of local oscillators found in computer clocks. It is also inconsistent with observations I have made of computer clocks; such observations are presented in Chapter 4.

Nevertheless, the notion that poffset should be small is sound — this corresponds to the intuitive notion that when a well-synchronized clock makes noisy measurements most of them are small. What is needed is to express carefully the notion that the local oscillator mostly behaves like an oscillator should.

### 2.5.3 Well-founded model of local oscillator

I consider models of the local oscillator based on physical principles. I assume that the local oscillator may be characterized by a phase, frequency error, and optionally an aging rate. The phase is the initial error in time: $\texttt{ttime}(t_0) - \texttt{utime}(t_0)$. The frequency error is the difference (expressed as a pure ratio, or in ppm), between the correct frequency, the dimensionless quantity 1, and the local oscillator's actual frequency:

$$1 - \frac{\texttt{utime}(t_2) - \texttt{utime}(t_1)}{\texttt{ttime}(t_2) - \texttt{ttime}(t_1)}$$

Thus a positive frequency error denotes a slow oscillator. The aging rate, usually expressed in ppm/day, is the time derivative of frequency error. An alternate way of expressing this model is to provide an expression for true time in terms of uncorrected time and the parameters making up the model:

$$\texttt{ttime}(t) = \texttt{utime}(t) + \left(p + f(t - t_0) + \frac{a}{2}(t - t_0)^2\right)$$

Aging is sometimes modeled and sometimes not; I will refer to the "linear model" and the "aging model" or "quadratic model" as appropriate. At times in this work I choose a linear model because I believe that more accurate results will be obtained with aging assumed zero. This issue is discussed in Section 3.6.

The phase component of the model is entirely straightforward — the counter simply has a particular value at the beginning of the region of interest ($t_0$). The frequency error component is also straightforward — quartz crystal oscillators oscillate at a particular frequency that depends on the precise details of their construction ([GB85], [Par83]). The frequency error is simply the difference between the actual frequency and the desired frequency.

Aging, the time derivative of frequency, is a well-known phenomenon in quartz crystal oscillators. According to Benjamin Parzen ([Par83], p. 109), aging has several underlying physical causes, such as redistribution of contamination within the crystal enclosure, slow leaks in the enclosure, and various stresses that are gradually relieved.

I do not model higher-order derivatives (thus avoiding the issue of whether aging is in fact a linear rate of change of frequency or a more complex process). I also do not model changes due to temperature changes, shock, vibration, power cycling, or other such causes. While the simple model presented here is not always completely accurate, it is simple enough to use and contains nothing ad hoc. I argue that it is an appropriate model in that it is well-grounded and allows me to achieve useful results.

Given the model of a local oscillator consisting of phase, frequency error, and aging rate, I develop a method of computing predicted time. Since I have assumed that the local oscillator can be characterized by a small set of parameters, I estimate those parameters in such a way as to make the resulting values of `poffset` small. I choose the sum of the squares of the resulting values of `poffset` as the quantity to be minimized; this is precisely the least-squares technique common in the surveying community.

This method is not completely reasonable either, as it assumes that the local oscillator may be perfectly characterized by these three parameters. As stated above, the model does not account for certain behaviors, such as those due to temperature changes or various other effects. I will defer addressing the issue of such unmodeled changes until later.

### 2.5.4 Estimation of parameters

Recall that the model of local oscillator behavior is that it may be characterized by a phase, frequency error, and aging rate:

$$\mathtt{ttime}(t) = \mathtt{utime}_{\mathtt{lcl}}(t) + \left(p + f(t - t_0) + \frac{a}{2}(t - t_0)^2\right)$$

Incorporating the assumption that the remote clock has the right time ($\mathtt{atime}_{\mathtt{rem}} = \mathtt{ttime}$), and rearranging terms:

$$\mathtt{atime}_{\mathtt{rem}}(t) - \mathtt{utime}_{\mathtt{lcl}}(t) = p + f(t - t_0) + \frac{a}{2}(t - t_0)^2$$

Then, incorporate the definition of `uoffset`:

$$\mathtt{uoffset} = \mathtt{atime}_{\mathtt{rem}} - \mathtt{utime}_{\mathtt{lcl}} + \mathtt{noise}$$

$$\mathtt{uoffset}(t) - \mathtt{noise}(t) = p + f(t - t_0) + \frac{a}{2}(t - t_0)^2$$

Thus, according to our model and our assumption that the remote clock is correct, values of uoffset differ only by noise from the value predicted by the model.

I consider a particular *predicted* time resulting from estimating phase, frequency error, and aging according to our model, and denote the estimates $\hat{p}$, $\hat{f}$, and $\hat{a}$:

$$\texttt{ptime}(t) = \texttt{utime}(t) + \left(\hat{p} + \hat{f}(t - t_0) + \frac{\hat{a}}{2}(t - t_0)^2\right)$$

From this I obtain expressions for predrsadj and poffset:

$$\texttt{predrsadj}(t) = \hat{p} + \hat{f}(t - t_0) + \frac{\hat{a}}{2}(t - t_0)^2$$

$$\texttt{poffset}(t) = \texttt{uoffset}(t) - \left(\hat{p} + \hat{f}(t - t_0) + \frac{\hat{a}}{2}(t - t_0)^2\right)$$

As before, I wish poffset to be small, expressing the notion that the offset measurements are generally correct. Now, however, I have sharply constrained our model of the local oscillator, allowing only 3 degrees of freedom rather than arbitrary behavior. I again use least-squares and choose $\hat{p}$, $\hat{f}$, and $\hat{a}$ to minimize

$$\sum_t \texttt{poffset}(t)^2$$

After choosing parameters, I have specified predrsadj. From this I can compute poffset and rsadjresid, according to the definitions.

## 2.6 Interpretation of poffset and rsadjresid

When analyzing the behavior of a synchronization scheme, the local oscillator, the network, or a remote clock, it is helpful to examine graphs of aoffset, poffset, and rsadjresid. Separating network and remote clock behavior is beyond the scope of this chapter and will be discussed later. Given the concepts of local oscillator, system clock, remote clocks, and networks, we can give descriptions for the various quantities computed:

aoffset a view of a remote clock, through a network, with respect to the system clock.

uoffset a view of a remote clock, through a network, with respect to the local oscillator.

poffset a view of a remote clock, through a network, with respect to the local oscillator together with its estimated parameters.

rsadj a view of the system clock, with respect to the local oscillator.

rsadjresid a view of the system clock, with respect to the local oscillator together with its estimated parameters.

These descriptions are of course not the definitions, but are helpful in understanding what the quantities represent.

Consider analyzing the behavior of the local oscillator (and not the synchronization scheme). Values of aoffset are not directly helpful, as they are contaminated by changes

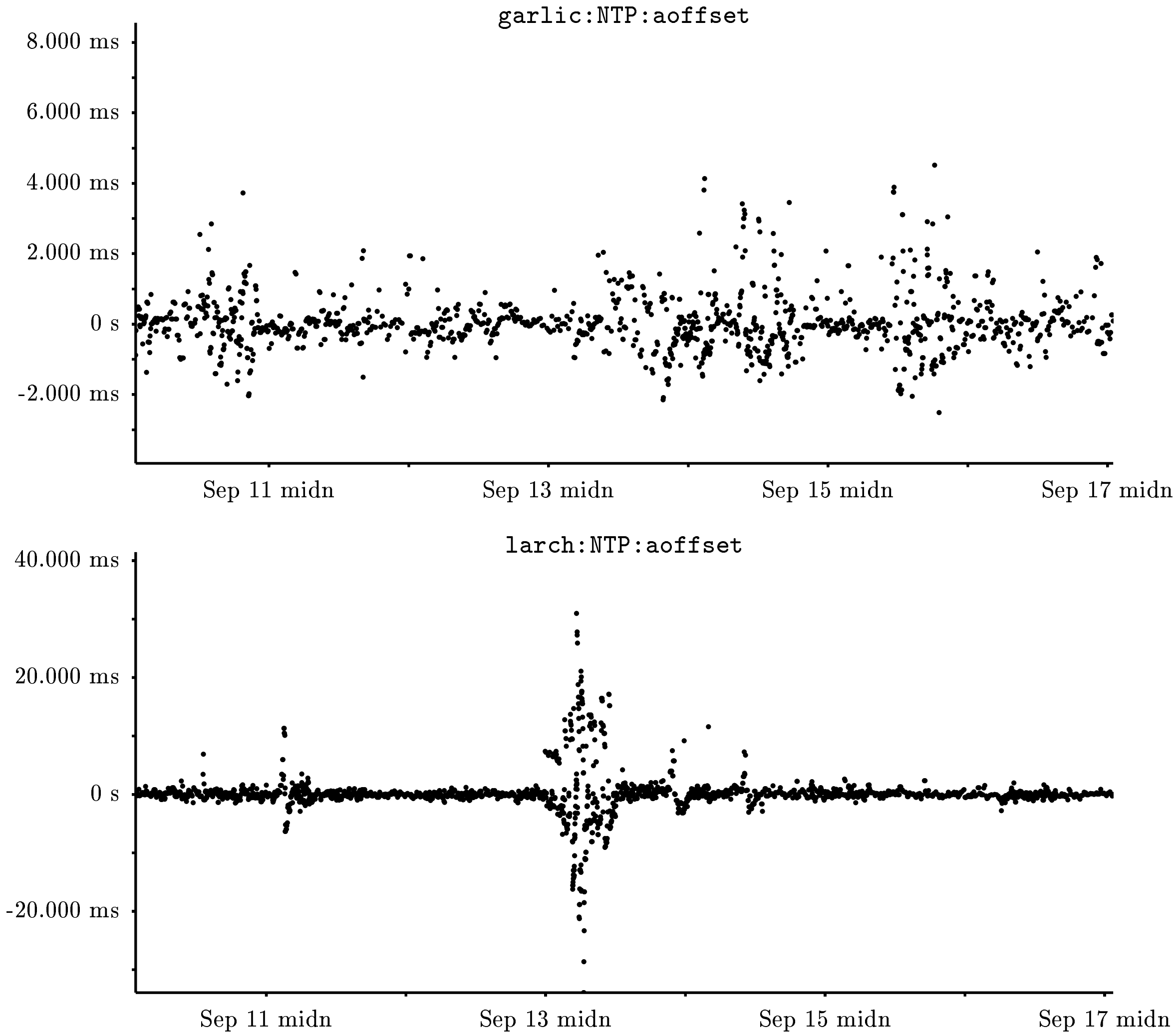



Figure 2.7: `aoffset` vs. time for garlic, top, and larch, bottom.

made to the system clock. Figure 2.7 shows actual offsets observed by two computers. Values of `uoffset`, while relevant, are not directly useful as the interesting features are obscured (as in Figure 2.3). Examining a graph of `poffset`, however, is very revealing. Figure 2.8 shows graphs of `poffset` covering the same time period for two computers. The local oscillator of the first computer, garlic, was at the time an EG&G TCXO (temperature-compensated crystal oscillator). The local oscillator of the second computer, larch, was the oscillator supplied by the manufacturer. There are two easily identifiable reasons that values of `poffset` are not zero — the model is not appropriate, and the offset measurements are noisy. In the first case, the values of `poffset` — the difference between observed behavior and predicted behavior from the estimation — are small, and no structure is apparent. I claim that the model fits the local oscillator fairly well in this case, and that the major cause of non-zero values of `poffset` is network-induced noise. In the second case, the data appear to be distributed about a wavy line rather than about zero. While the effects of network

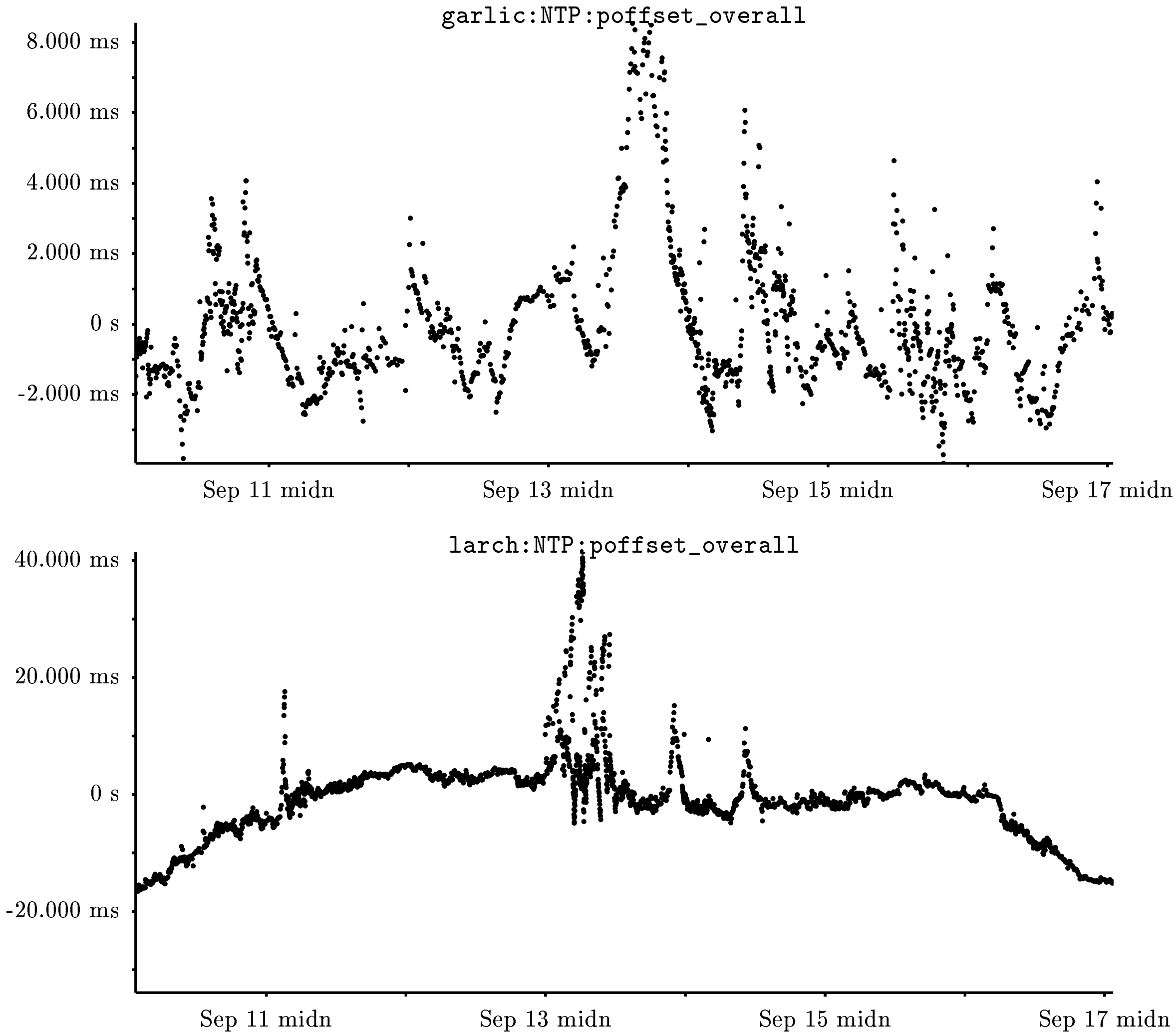



Figure 2.8: `poffset` vs. time for garlic, top, and larch, bottom.

noise can be clearly seen, there is a much more distinct long-term trend. In this case, I claim that the model does not fit the local oscillator very well because the computer's clock exhibited varying frequency error. Just as the slope of `uoffset` was explained as frequency error, the slope of portions of `poffset` is the difference between the actual frequency error and the estimated frequency error. The frequency error estimate can be considered to be an estimate of the average frequency error.

Figure 2.9 shows corresponding values of `rsadjresid` for the same computers. The values for garlic are small, indicating that the difference between what the synchronization algorithm did in fact do and what the estimated parameters say it *should* have done is fairly small, with a notable exception late in the day on September 14. Note that this feature is not apparent in Figure 2.7 — the `rsadj` technique has separated the behavior of the network and remote clocks from the behavior of the synchronization algorithm. The values for larch are larger, and different in character. While there appear several instances

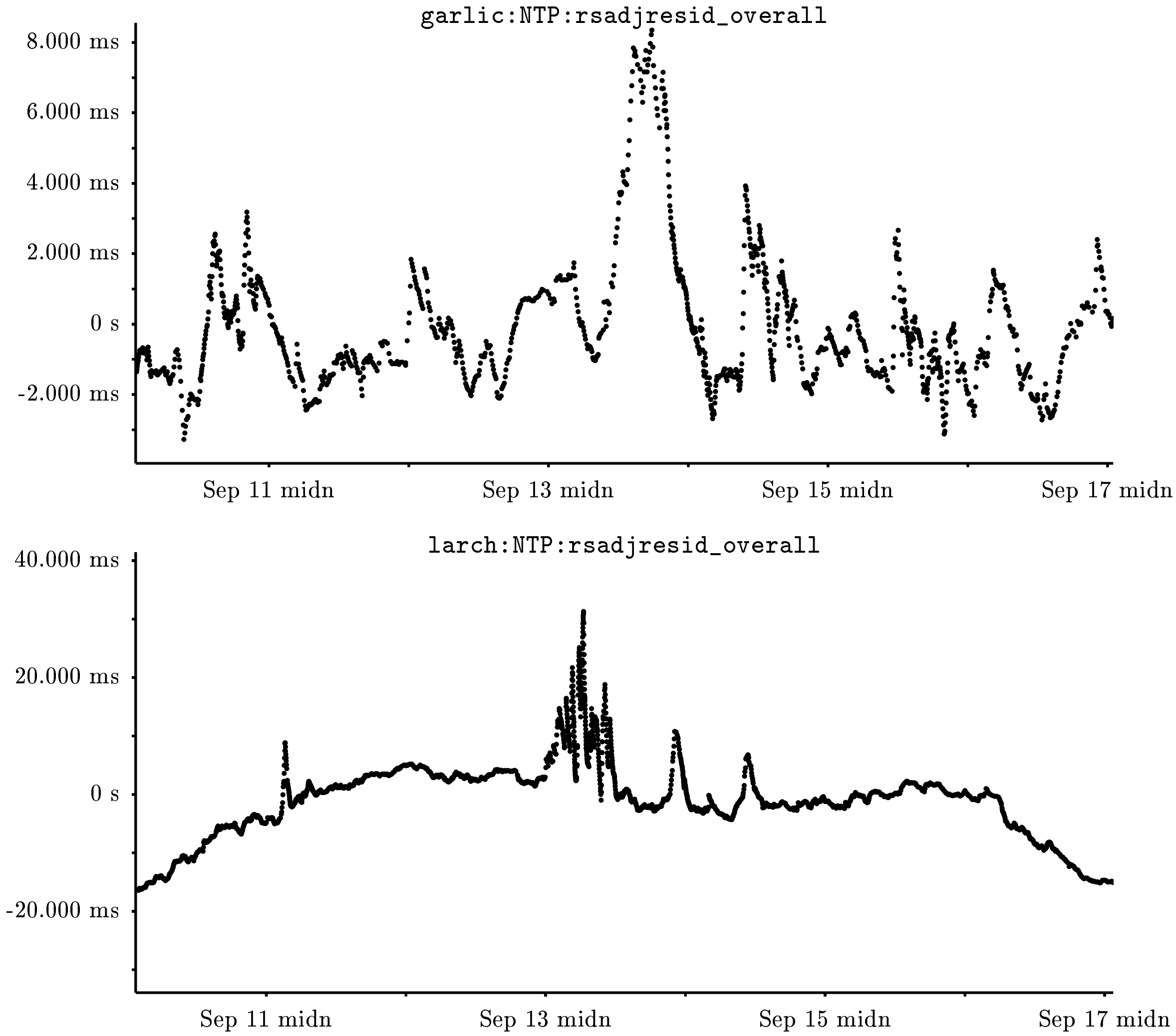



Figure 2.9: `rsadjresid` vs. time for garlic, top, and larch, bottom.

where the synchronization algorithm departed from the wavy line, once by 30 ms, most departures are small. In this case, even though many values of `rsadjresid` are significantly non-zero, the long-term structure of the values argues that systematic long-term changes and not short-term network delay-induced noise are responsible for the non-zero values. In this case NTP, the actual synchronization algorithm being run, was correct except for a small number of time periods.

# Chapter 3
# Computation of Predicted Time

This chapter describes the computation of predicted time, which is the process of computing estimates of the parameters of the local oscillator. The techniques for computing predicted time are extensions of those already presented. As a Time Surveyor, I need more complex methods than those shown already in order to deal with the real behaviors possible with local oscillators, networks, and remote clocks. I must identify observations that are *mistakes* so that they may be removed from consideration. Due to the nature of time, retaking observations is not possible. A further complication in Time Surveying is that I must decide when the basic assumption about local oscillator behavior — that it can characterized by phase, frequency error and aging — does not hold. I present mechanisms for making this decision.

There are several sources of mistakes in an environment in which clocks are being synchronized. One is excessive network delay. While one could consider the observations resulting from high-delay packets to be random errors, I have not found the distribution of delays and resulting offset values to be Gaussian. I choose instead to treat very high-delay observations as mistakes, and consider the remaining observations to have random errors. Another source of mistakes is occasional errors in timekeeping by a remote clock. I present a technique to identify time periods when a remote clock is not self-consistent and to remove from consideration observations made during those time periods.

This chapter presents several versions of the computation of predicted time, termed *overall*, *piecewise*, and *integrated*. Each of these versions has a distinct purpose. The overall computation is useful for obtaining a broad view of the behavior of the local oscillator, the network and remote clocks. It is also useful for obtaining a broad view of the behavior of a synchronization algorithm. This computation does not declare any offset measurements to be mistakes, and assumes that the local oscillator can be characterized by a single set of estimates over the entire time period for which data are available. In the typical case where a single set of estimates may usefully, if not precisely, describe the behavior of the local oscillator, the effects of both minor and substantial unmodeled changes in the local oscillator can be seen. The overall computation is also helpful in cases where anomalous behavior of some component confounds the more complex versions of the computation.

The piecewise computation of predicted time has as its goal the accurate characterization of the behavior of the local oscillator as long as the behavior of the remote clocks as observed via the network is not unreasonably faulty. The computation can tolerate some incorrect behavior, but not behavior that is *usually* incorrect. In the piecewise computation, some observations will be declared to be mistakes, and will be marked *invalid*, because they have high delays, or because they are inconsistent with other nearby observations, for example. Rather than producing a single set of estimates of local oscillator parameters, the piecewise computation produces a set of estimates and time periods over which those estimates are valid. Changes in frequency of the local oscillator are accurately identified, given remote clock and network behavior that is not grossly erroneous. In the case of erroneous remote clock behavior, the behavior of the remote clock is misinterpreted as changes in the local oscillator. Given that one knows from examining non-erroneous remote clocks that such changes did not in fact occur, such misinterpretation can be usefully viewed as an analysis of apparent frequency changes of the remote clock.

The integrated computation of predicted time, rather than trying to provide an accurate characterization of the local oscillator over the entire time period for which data are available, attempts to provide a characterization that is very accurate over the immediate past. The computation of integrated predicted time uses data from multiple remote clocks, and only considers recent data. The notion of recent is derived for each clock by balancing the expected errors due to network delay variance and those due to unmodeled changes in the local oscillator. For remote clocks with greater network delay variance, more changes in the local oscillator are tolerated. While such a characterization is inspired by the desire to synchronize the clock rather than merely examine it, integrated predicted time is useful for analyzing the behavior of remote clocks because a view of a remote clock believed to be erroneous may be obtained relative to local oscillator parameters derived from many remote clocks. Such a view shows the behavior of a particular remote clock without having to assume that the remote clock in question is usually correct.

I compute predicted time both as an aid in analyzing synchronization performance, and as a mechanism to improve future performance. These are different goals, and I compute predicted time in different ways to achieve each goal. This chapter describes the computation of predicted time for analysis of synchronization. In Chapter 5, I describe small changes to the integrated computation of predicted time in order to make it suitable for synchronizing the system clock.

I will use the following terms:

**interval** a set of times described by a begin time and an end time, containing the endpoints and all times in between.

**fit** a set of local oscillator parameter estimates $\hat{p}$, $\hat{f}$ and $\hat{a}$ and an interval. A fit may optionally contain multiple phases to describe different remote clocks or different periods of continuous operation of the computer; it may be desired to characterize periods of time before and after a given crash of the operating system with the same frequency estimate.

**point** an offset observation, i.e., a time and `uoffset`: $(t_i, \mathtt{uoffset}_i)$

**invalid point** an observation which has been marked invalid by a processing step.

**residual** the difference between a `uoffset` and the value predicted by the fit in question, i.e. `poffset` (within the context of a particular fit). When I refer to a residual as being "large," I mean large in absolute value.

**rms error** the square root of the mean of the squares of the residuals of all valid points in an interval with respect to the estimates of a given fit. The term rms error, when applied to a set of observations without reference to a particular interval or estimates, will mean the rms error with respect to the interval covering all the offset observations and the estimates resulting from that set of observations.

As in Section 2.5, I compute `predrsadj` rather than `ptime` itself. Each computation of predicted time is a function taking as input `uoffset` and producing as output `predrsadj` and a label for each observation stating whether it is invalid, and if so, why. Rather than specifying `predrsadj` directly, the output of the computation is given as a set of fits, i.e. time intervals and local oscillator parameters: $((t_{\mathtt{begin}},\ t_{\mathtt{end}}),\ (\hat{p}, \hat{f}, \hat{a}))$. In addition the rms error of each fit is computed, although this is not strictly speaking an output of the computation.

Many of the steps involved in the computation of the piecewise and integrated versions of predicted time involve configurable constants. While I have made every effort to choose reasonable constants for well-grounded reasons, I realize that some of the values are tied to constants in the NTP implementation itself. These configurable constants will be identified, and the "current values" for the algorithm given. Thus, I am really describing a parameterized family of computations in the context of a particular choice of parameters. The application of these computations to problems far removed from today's clocks, packet networks and synchronization schemes can possibly require different values of the constants, and the discussion here should give guidance on choosing those constants based on an understanding of the systems at hand.

## 3.1 NTP's system offset versus raw offsets

In the previous chapter, I used as input values of `uoffset`, or *uncorrected offsets*. The offsets used were NTP's *system offset*, the output of NTP's clock selection and combining algorithms. The choice of NTP's system offset as the input to the `rsadj` technique has advantages and disadvantages. Perhaps the most important advantage is that it is the obvious choice if only one offset can be observed. Consider the task of producing only a few summary graphs of a computer's synchronization performance. While showing values of `aoffset` and `poffset` to each remote clock would provide much information, it is not a compact summary of the synchronization behavior. The use of NTP's system offset provides such a summary. I also consider the use of *raw* offset observations, which are the results of NTP's offset measurement operation without further processing.

I compute the overall and piecewise version of predicted time on the NTP system offsets and separately for the raw observations from each remote clock. The integrated version is based on raw observations from multiple remote clocks.

### 3.1.1 NTP's system offset

NTP's clock selection algorithms remove from consideration remote clocks that are not self-consistent. They also cast out remote clocks that disagree with the consensus value of time among all remote clocks. Then, they select a small number of the remaining clocks with low variance and combine their offsets to form the system offset. Because of this selection and combining process, the system offset is almost never grossly wrong; if one clock returns time that differs greatly from the others, it will be marked as incorrect and its value will not be used.

A serious problem with the system offset is variation in the selection process; as raw offset measurements are made of remote clocks, the selection and combining processes produce

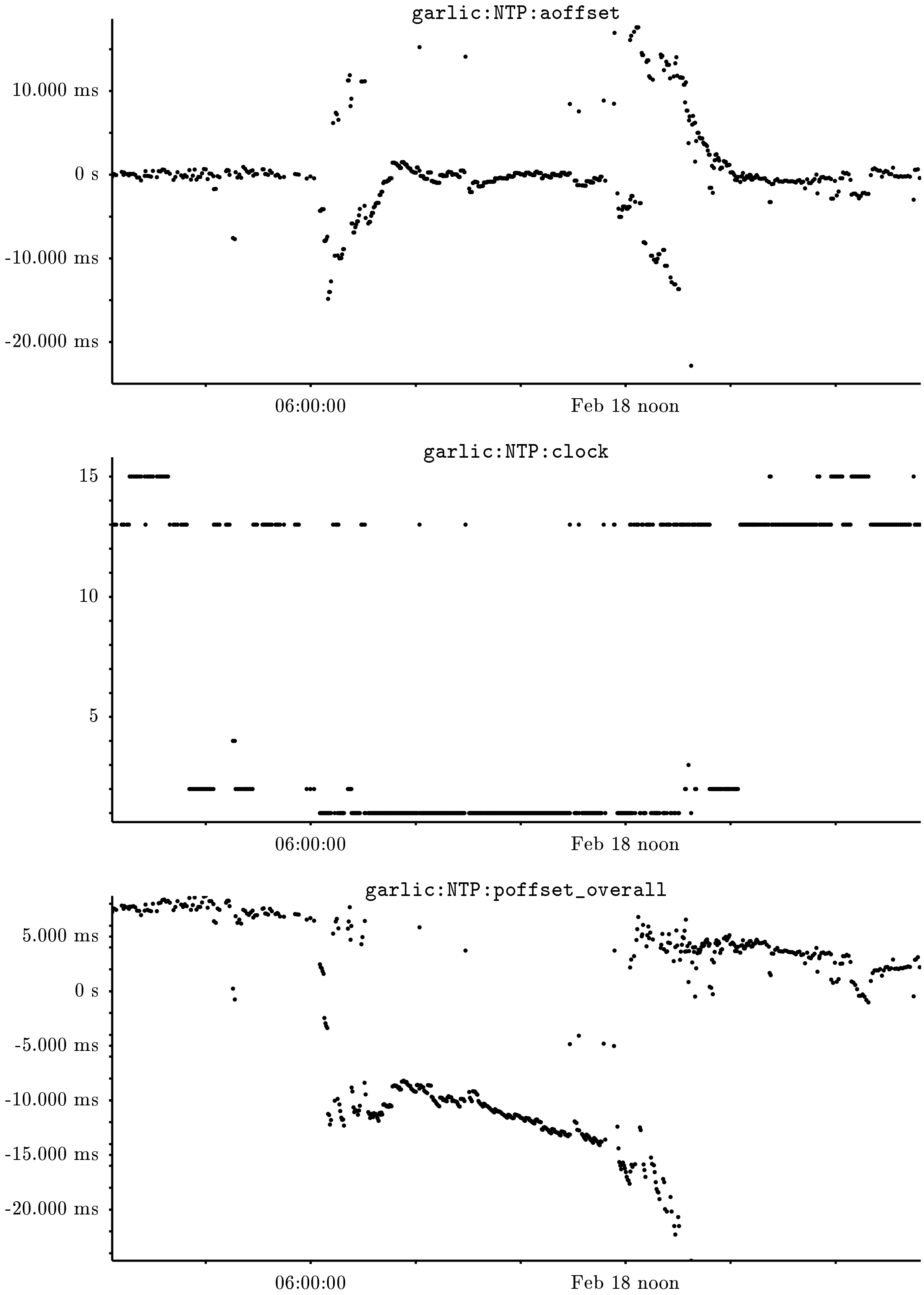



Figure 3.1: NTP's actual system offset, above, index of the currently-selected remote clock, middle, and predicted system offset, below, vs. time.

different results. The problem is that small variations in offset values can and do change the ordering of clocks in the selection and combining process, causing the final result to be more closely steered to the offset values of different clocks at different times. Figure 3.1 shows an example of such behavior. This is a far greater problem when using the `rsadj` technique than with normal NTP synchronization, because much longer time periods are examined. Over these time periods, differences of a few milliseconds in systematic offset between remote clocks can be observed; such differences are often obscured by noise in the shorter time scales over which NTP examines offset data.

Another problem with system offsets which arises only with the use of the `rsadj` technique is that while system offsets are produced at regular intervals (corresponding to the PLL update rate), the system offset only changes when a new offset sample from one of the clocks used in the combining arrival arrives and that new sample is the lowest-delay offset sample currently kept for that remote clock.[1] This is intentional in the NTP design; the intent is to use only the lowest-delay recent offset sample because high-delay offset samples are likely to have greater errors. However, when using the `rsadj` technique, this causes low-delay offset samples to be weighted according to the length of time until the arrival of the next low-delay sample. Since the `rsadj` technique attempts to treat offset samples without respect to the sequence in which they were obtained, this effect is undesirable. Another undesirable effect is that the time logged with the sample is decoupled from the actual time the observations were made.

### 3.1.2 Raw offset data

For the reasons given above, I also use raw offset data for each remote clock. Raw offset data are the output of NTP's basic offset measurement operation, and consist of the time at which the measurement was made, the name of the remote clock, the offset observed, and the total delay observed. The value of `rsadj` is also recorded, so that a value of `uoffset` can be computed from the recorded `aoffset`. This approach provides the synchronization analysis tools with the raw data obtained by NTP while attempting synchronization. With raw offset data, I can analyze the behavior of remote clocks not selected by NTP, and analyze the behavior of the local oscillator with respect to those remote clocks. I may use longer time periods than the 8-sample filter used by NTP in order to determine which samples are truly low-delay. I may analyze each clock separately, and then determine the systematic offsets among various remote clocks.

The modified daemon does not actually record all raw data. Only offset samples which at the time of measurement have the lowest delay of the eight most recent samples are recorded. This is done in order to reduce the volume of data recorded. This early pruning is acceptable because every offset measurement not recorded by the daemon has a higher delay than a recent measurement that was recorded. In later processing, the computation will discard approximately half of the recorded offset samples because they have high delays.

[1] I examined 3 days of typical data; 1 in 6 NTP offsets were the same value as the preceding one.

## 3.2 Computation of uncorrected offsets

The first step in the computation is to acquire all the offset observation data, both for NTP system offsets, henceforth called NTP data, and raw offset measurements, henceforth called raw data. The data are organized into several sets: one set for NTP data, and one set for raw data associated with each remote clock.

The next step is the computation of `uoffset` values. As stated in the previous chapter, $\texttt{uoffset} = \texttt{aoffset} + \texttt{rsadj}$. This computation is done for both NTP data and for each raw data set.

## 3.3 Overall predicted time

In Chapter 2, I presented a basic computation of predicted time. This computation took as input the `uoffset` values of NTP's system offsets, and produced a single set of oscillator parameters. The values of all offset observations were used, and each observation was weighted equally.

Here, I define the computation of overall predicted time, which is very similar to the computation presented previously. A value of overall predicted time is computed for NTP data (NTP's system offset), and each set of raw data (the set of observations for each remote clock). If there are observations from $n$ distinct remote clocks, $n+1$ values of overall predicted time will be computed.

While there are a number of problems with this scheme for computing predicted time, it is very valuable as a means of producing an overview of the data. One can examine the values of `poffset` produced by the overall computation and determine quickly whether the fit is reasonable or erroneous — a single grossly erroneous observation can greatly affect the estimation. One can also determine whether the assumption that the local oscillator can be characterized by one set of parameters over the time period is reasonable or not. In the case where this assumption does not hold, however, valuable insight into exactly *how* it does not hold can be gained; Figure 3.2 shows a case in which the frequency error of the local oscillator changed.

One complication relative to the previous presentation is that a real computer system loses power, is rebooted, or crashes from time to time, interrupting the running of the timekeeping daemon. When this occurs, the phase of the local oscillator is usually lost, as well as the value of `rsadj`. I term these distinct periods of continuous operation *runs*. While it is possible to design a system so that the phase of the local oscillator is preserved across these events (see Section 6.3.1), typical computer systems do not have this capability.

To deal with data covering multiple runs, the model of a local oscillator is extended to consist of a phase for each run, as well as frequency error and aging estimates. For example, given data covering runs 0, 1 and 2 (arbitrarily numbered for convenience), I would estimate $p_0$, $p_1$, $p_2$, $f$, and $a$. When calculating $\texttt{predrsadj}(t)$, the value is given by

$$\texttt{predrsadj}(t) = \hat{p}_0 + \hat{f}(t - t_0) + \frac{\hat{a}}{2}(t - t_0)^2$$

for times between the first and last times recorded for run 0, and similarly for the other runs.

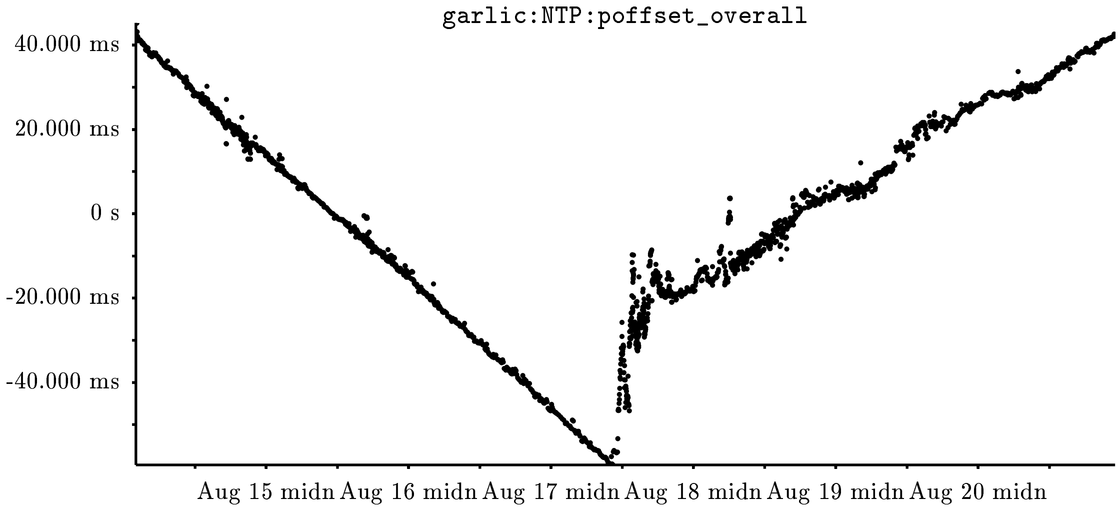



Figure 3.2: Values of `poffset` with respect to the overall computation for a time period during which the local oscillator was replaced. The author swapped oscillators on a running system by moving wires on a breadboard, causing a frequency error change.

The estimation process is straightforward given the addition of the concept of runs; a separate phase error $\hat{p}_i$ is estimated for each run $i$. For $n$ runs, the $n+2$ estimates $\hat{p}_i$, $\hat{f}$, and $\hat{a}$ are chosen so that for each observation $(t_{ij}, u_{ij})$, where $i$ ranges over the run numbers and $j$ ranges over the observation number within the run, the sum of the squares of the resulting values of `poffset` is minimized:

$$\sum_i \sum_j \left( u_{ij} - (\frac{\hat{a}}{2} t_{ij}^2 + \hat{f} t_{ij} + \hat{p}_i) \right)^2$$

The run mechanism allows for phase discontinuities, but does not introduce any notion of frequency changes across runs. While the temperature changes caused by losing and regaining power might affect the frequency and aging rate of a quartz crystal oscillator ([Par83]) I do not attempt to model any such changes. Figure 3.3 shows values of `poffset` for a time period containing two different runs. Note that the two trends evidenced by `poffset` values are discontinuous. There is no reason to expect them to be aligned, as the values of `poffset` are zero-mean for each run, and unmodeled errors cause the values of `poffset` to be non-zero.

The computation of overall predicted time does not mark any points invalid, and outputs one fit with an interval that covers all of the offset observations.

## 3.4 Piecewise predicted time

While the overall computation of predicted time attempted to produce predicted time in as simple a manner as possible, the computation of piecewise predicted time attempts to compute predicted time accurately. The piecewise computation marks some offset observations as invalid and therefore ignores their values. It also produces multiple fits, the

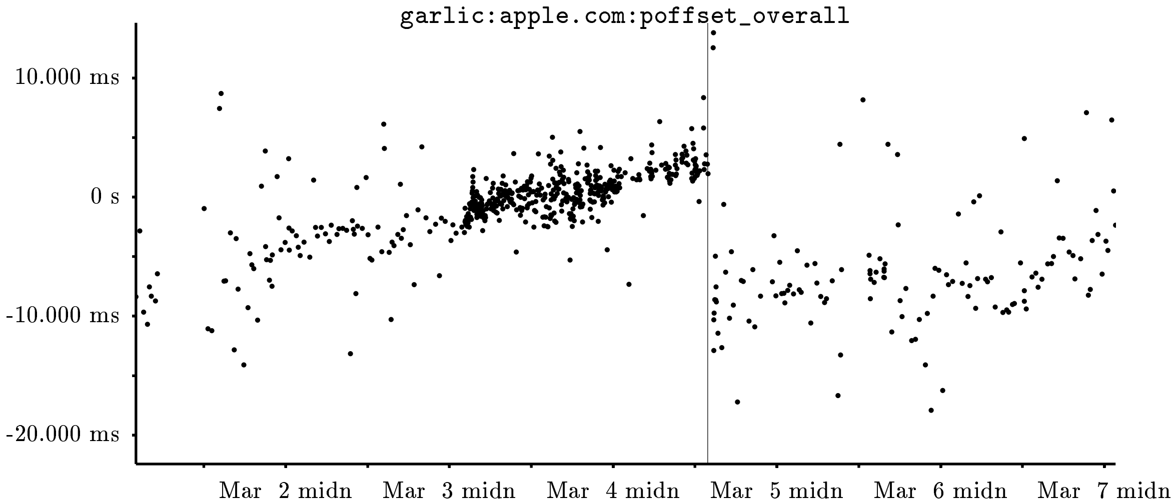



Figure 3.3: Values of `poffset` over two runs with respect to an estimation of a single frequency and two phases. The runs are separated by a vertical line.

intervals of which cover the entire time period of the input observations, and the estimates of which reasonably characterize the observations made during those intervals.

If one knew that a single set of parameters of the basic local oscillator model — phase, frequency error and aging rate — described the local oscillator over the entire time period covered by the given data, then one could simply estimate the local oscillator parameters, as I do in the overall fit. One does not usually know that this assumption is valid, however, and in many cases it is in fact not valid. There may be unmodeled changes in the local oscillator behavior, such as those due to temperature changes, shock, or even complete replacement of the oscillator.[2] Thus, I wish to determine intervals over which a set of parameters for the basic model reasonably characterizes the local oscillator's performance. There is no predetermined size of the intervals; an attempt is made to have the intervals be as large as possible and still have the estimates reasonably describe the behavior of the local oscillator.

To determine what is meant by "reasonably describe," I consider the various sources of error in the system, i.e., conditions which result in non-zero values of `poffset`, and thus non-zero rms errors. One obvious source is the offset errors resulting from network delay variance. Another is that the remote clock under consideration might not always have the correct time, i.e. $\texttt{atime}_{\texttt{rem}} \neq \texttt{ttime}$. A third is that the local oscillator might exhibit deviations from the model, such as a change in frequency over time. I deal with these errors in different ways in the computation of piecewise predicted time. The computation marks as invalid offset observations with very high delay, but some errors due to delay variance will remain. It attempts to identify periods in which the remote clocks behave as if their value of time was incorrect, and mark as invalid samples from those periods; this process is

[2]While complete replacement of an oscillator might at first seem odd, it is exactly the result of the practice, increasingly common in my research group, of considering a "computer" to be the disk on which the data of interest resides, and the processor merely an interchangeable appliance to be applied to this data. When a disk is moved between processors, it appears to the synchronization algorithm that the local oscillator has been replaced.

very successful for remote clocks that are usually not erroneous. The last identified error source, unmodeled behavior of the local oscillator, is mitigated by attempting to make it of the same magnitude as other error sources by controlling the size of the intervals over which estimates are made. The intervals are made as large as possible subject to the constraint that the rms error over the interval is no greater than a small multiple of the estimate of the average network-induced error.

The computation of piecewise predicted time consists of several distinct steps, listed briefly here and discussed in detail in later sections.

1. Observations belonging to a run containing fewer than a small number of points or covering less than a small time period are marked invalid, unless the run in question is the most recent run observed.

2. The delay characteristics of the offset observations are examined, and high-delay points are marked invalid.

3. The observations are divided into a number of segments, each covering up to typically 4 hours.

4. The median value of the rms error for the segments is calculated. All points belonging to a segment with an rms error at least twice the median error are marked invalid.

5. For each segment, all points with a residual greater than twice the rms error for the segment are marked invalid.

6. Adjacent segments are repeatedly examined, and a measure of how well they may be described by a single set of parameters is computed. If this "measure of similarity" is sufficiently good, they are combined, so that the two original segments no longer exist, and the list of segments contains in their place a new segment containing all the points contained in the original segments.

7. For each segment remaining on the list, all points with a residual greater than twice the rms error for the segment are marked invalid.

8. For each segment, the estimates are examined. If an estimate does not contain a phase value for a given run, the phase for that run is estimated from all observations for that run and the frequency error and aging estimates already computed.

### 3.4.1 Trimming: Invalidation of *a priori* likely incorrect data

Observations belonging to a run containing fewer than a small number of points or covering less than a small time period are marked invalid, unless the run in question is the most recent run observed. The current value for "small number of points" is 20, and the current value for "small period of time" is 2 hours. The motivation is that normal operation involves observing a remote clock over a reasonably long period of time.

It seems likely that frequent rebooting of a computer may be caused by some sort of problem or anomalous condition in the operating environment. Thus it seems plausible to guess that data gathered while the computer was only able to run for 1 hour, for example,

```
TRIM: INVALIDATED RUN 0 POINTS 19 TIME 36864 (0-18)
TRIM: ACCEPTED RUN 1 POINTS 213 TIME 219728 (19-231)
```

Figure 3.4: Example output of "trim" processing. Run 0 covered 36 864 s, or over 10 hours, but only contained 19 offset measurements, and thus those points were marked invalid.

may be erroneous. Since the computation estimates frequency, runs of this length are of questionable usefulness given nearby runs of longer duration. Several of the later steps involve statistical information over all of the data. The removing of isolated data points helps to avoid misapplication of such statistical characterizations to data in cases where the short run was in fact correlated with other changes.

The "short run" test is not applied to observations from the most recently observed run for two reasons. One is that a short recent run, for example 1.5 hours, is only really "short" on the side going back in time to its beginning. While only 1.5 hours have transpired, the run is not terminated — it is merely that no information is available concerning when it ended, which is quite different from having information that it did end. Also, the trimming scheme is designed to satisfy the needs of computing predicted time to proactively synchronize the system clock. For this computation it is necessary that data from the most recent run be kept if the predicted time to be used for future timekeeping is to be relevant and accurate. This issue is discussed further in Chapter 5.

### 3.4.2 Invalidation of high-delay observations

The next step in the computation is "the delay characteristics of the offset observations are examined, and observations having high delay are marked invalid." This process is complex, as adverse affects due to a small number of erroneous points must be avoided, and the computation must cope with changes in the network configuration that cause persistent changes in delay characteristics. In a sense the computation selects for use low-delay observations, rather than performing the equivalent process of marking invalid high-delay observations.

This step is only performed on raw data. It is not performed on NTP data because NTP data combines offsets from several clocks, so it is not clear how a delay should be assigned to each NTP offset data point. Also, an increase in the delay value of an NTP system offset observation because of a change in clock selection does not necessarily indicate anything at all about the quality of the data.

Recall that earlier I stated the modified NTP daemon only logs offset observations from each remote clock if the observation has the smallest delay of it and the 7 previous attempts to make offset observations. Thus, I consider the first step to be marking invalid all offset observations with a delay greater than any of the 7 previous attempts, as well as all unsuccessful attempts, meaning those in which no response was received.

When the NTP algorithms are being run on a computer, data from the future is not available, and thus NTP's method of selecting low-delay observations takes into account only previous measurements. The `rsadj` technique has the advantage of being able to examine offset observations occurring later in time than a particular observation when determining whether that observation is high delay. For example, the computation could mark invalid any observation with a delay greater than any of the *following* 7 observations, rather than the previous 7 observations.

For the remote clock under consideration, various order statistics of the delays of the observations are computed. Figure 3.5 shows sample output of the program implementing this computation; in this case observations spanning about one month and consisting of 14591 offset measurements of mizbeaver.udel.edu made by garlic are being examined. This delay distribution is fairly typical of a well-behaved network. The delay values are clustered near a "baseline" delay, and even the 75th percentile is near the base delay. Higher percentiles, however, have more significantly elevated values. In many of the graphs in this document, observations that have been marked as high-delay will be green.

Given statistics about the distribution of delay values, I must now decide which of the points will be accepted as "low delay," and which will be marked invalid. One extreme, choosing the 1% or so points with the lowest delay, for example points with delay less than 27.94 ms, is not appealing for several reasons, and it is instructive to examine them. One is that most of the data would be discarded, leaving only 146 or so points over the month-long interval in this example. Another, perhaps more serious problem, is that it is possible that all of the recent data points will be above the first percentile delay. While variance in network delay is a major source of errors in offset values, there are other significant sources such as errors in the time of the remote system and unmodeled changes in the local oscillator. The use of more rather than fewer data points helps to mitigate the effects of these errors, and excluding all recent points is unwise.

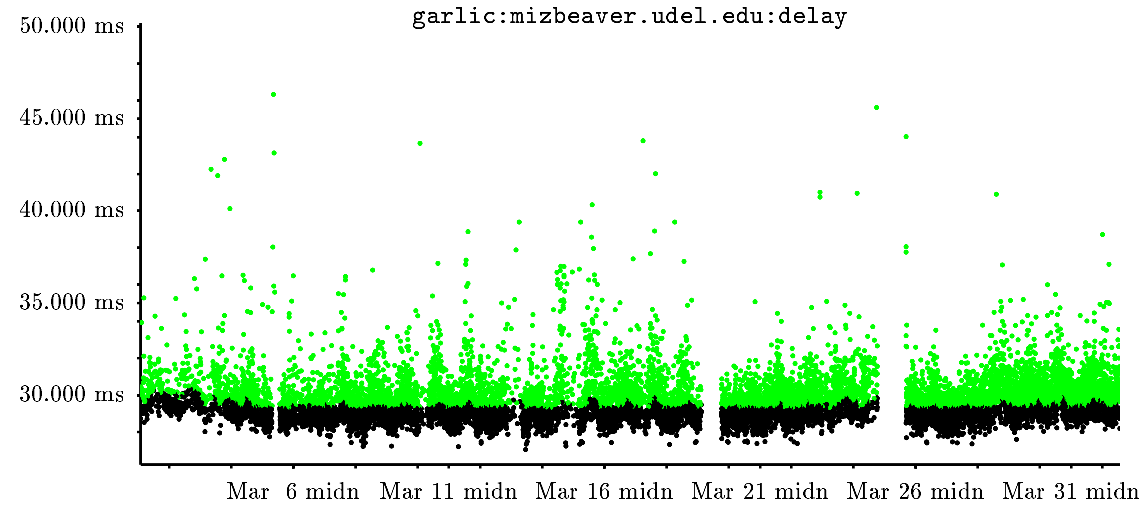
Even a scheme that selected the observations with delays below the 50th percentile could have similar problems. Consider the case in which a network was reconfigured, increasing the values of delay. In such a case, the simple scheme just described might still mark as invalid all of the recent observations. Thus, I desire that the chosen method adapt to changes in base delay such as these.

I adopt as the intent of the low-delay selection process to mark as invalid observations that have delays significantly higher than other points nearby in time. Because almost all observations encounter some additional delay, and because there are other sources of error, it is undesirable to discard all observations in the neighborhood of a low-delay observation. The selection algorithm is designed to avoid marking as high-delay observations whose delay is only somewhat higher than the current nearby low-delay point. The notion of "nearby in time" is intended to reflect the belief that while an observation with a significantly higher delay than one 5 minutes ago should be marked as having high delay, an observation that has a high delay only when compared to observations 24 hours or more distant should not be marked invalid.

An additional consideration in designing the low-delay selection algorithm is a desire for its time complexity to be linear in the number of observations rather than quadratic. Also, stability considerations which will be discussed in Chapter 5 make it desirable that the addition of new observations (later in time) not overly affect which of the previous observations are selected.

Since I would like the low-delay selection algorithm to work over a wide range of network behavior, I must avoid making quantitative assumptions about "normal" delay behavior. I do introduce a configurable constant to represent the amount time over which it is reason-

```
LOW DELAY  27.039 ms  HIGH DELAY 376.755 ms
1%       5%       10%      25%      50%      75%      85%      90%      95%      99%
27.94    28.30    28.52    28.84    29.31    30.03    30.50    30.90    31.62    34.29
p_base 28.305 p_add 29.312 p_mul 30.029 add 1.007 mul 0.060908
DELAY: ORIG 14591 OK 8246 INVALID 6345 (0.435 invalidated)
```



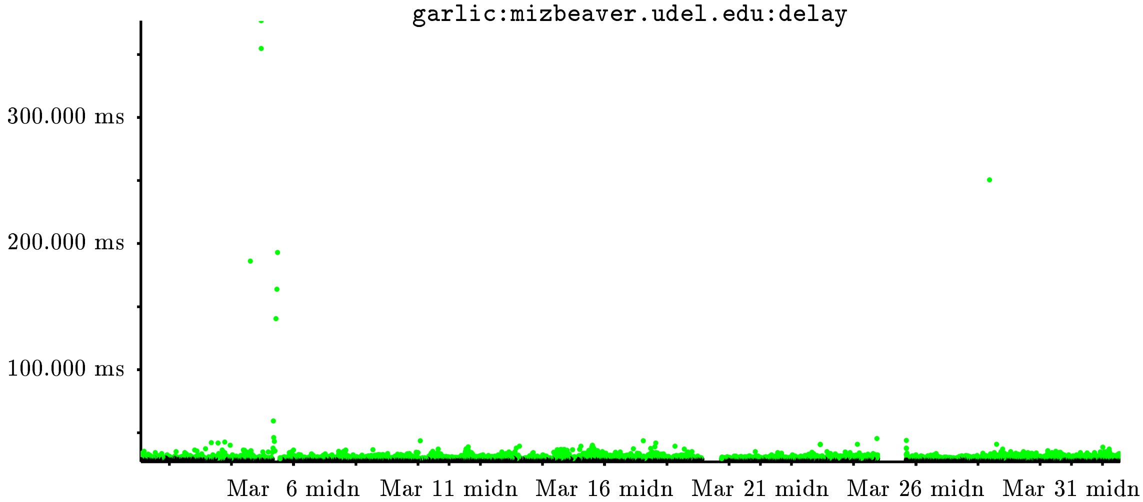




Figure 3.5: Sample output of the low-delay selection algorithm (all delays given in ms), and actual observed delay values, middle, and below. The bottom graph shows the entire range of delay values, while the middle graph shows more detail in the region of interest. The green points have been marked invalid because they are high delay.

able to discount a low-delay observation. I choose 12 hours as the value of this constant, attempting to represent the time scale of the diurnal variations in network behavior, presumably due to congestion induced by human behavior. I would like the computation to be willing to discard observations affected by daytime congestion, but do not want to be willing to discard several days' worth of data.

I also desire to avoid using any information about or from the extreme tails of the delay distribution, because in addition to normal variations in delay, arbitrary delay values may occur. Unless there are a large number of such errors, the errors will not greatly affect the value of the 50th percentile of delay, for example. I have recorded offset observations with negative delay; this is likely due to a hardware or software malfunction. While I saw only a few such observations, the presence of even one (incorrect) negative-delay observation would confound a process making calculations based on the lowest-delay observation.

The scheme I have chosen to select low-delay observations keeps observations which are locally low-delay, and those which have only somewhat more delay than the nearby low-delay observations. When this reference low-delay observation is more distant in time, an even greater increase is allowed. In order to realize the above scheme, the delays at the 5th, 50th, and 75th percentiles are computed, and termed $p_{\mathrm{base}}$, $p_{\mathrm{add}}$, and $p_{\mathrm{mul}}$. I let

$$\mathrm{add} = p_{\mathrm{add}} - p_{\mathrm{base}}$$

and let

$$\mathrm{mul} = \frac{p_{\mathrm{mul}} - p_{\mathrm{base}}}{p_{\mathrm{base}}}$$

Each data point is examined, and if the delay for that observations is greater than or equal to $p_{\mathrm{base}}$, all other points with excessively high delay relative to the point being examined are discarded. A point $(t_j, d_j)$ (ignoring the offset value) has excessively high delay relative to a point $(t_i, d_i)$ if

$$d_j > d_i \left(1 + \mathrm{mul}\frac{|t_j - t_i|}{12\ \mathrm{h}}\right) + \mathrm{add}$$

The actual algorithm used to select points for discarding produces the same results as the algorithm just described, but has time complexity linear in the number of observations to be examined. Rather than considering each observation as a potential low-delay point, and marking invalid all other observations which are high-delay relative to it, the observations are considered sequentially, exploiting the linearity of the expression defining the notion of high-delay. Each point $(t_i, d_i)$ is examined in time order, starting from the oldest, and compared to the current lowest-delay point $(t_l, d_l)$. The "aged" delay value $d_a$ of the low-delay point is computed as

$$d_a = d_l \left(1 + \mathrm{mul}\frac{|t_j - t_i|}{12h}\right)$$

If the delay of the current point is below $p_{\mathrm{base}}$, it is not discarded, but does not become the current lowest-delay point. If the delay of the current point is lower than the "aged" delay value, the current point becomes the lowest-delay point, and is not discarded. If the delay of the current point is greater than $d_a + \mathrm{add}$, the current point is discarded. When all points have been examined in the forward direction, the entire set of data is then examined in the reverse direction.

```
LOW DELAY  71.213 ms  HIGH DELAY 136.612 ms
1%      5%      10%     25%     50%     75%     85%     90%     95%     99%
86.30   86.99   87.39   88.10   89.16   91.13   93.51   95.57   101.20  112.92
p_base 86.990 p_add 89.157 p_mul 91.125 add 2.167 mul 0.047534
DELAY: ORIG 747 OK 447 INVALID 300 (0.402 invalidated)
```

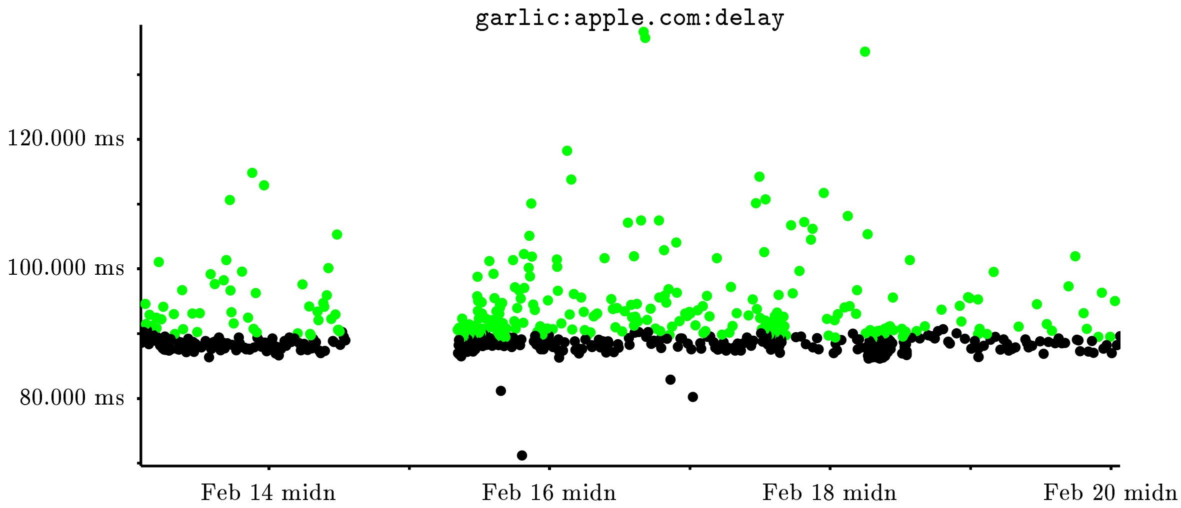



Figure 3.6: Delay values and statistical information for observations of apple.com as observed from garlic. The green points have been marked invalid because they are high delay.

Figure 3.6 shows delay values and statistics exhibiting anomalous low-delay points, and shows the results of the low-delay selection. Note that observations in the neighborhood of the anomalous observations are not marked invalid. Figure 3.7 shows delay values and statistics for a case where the base delay changed, and shows the results of the low-delay selection. Note that it is not the case that all observations in the region of higher delay were marked invalid.

### Meaning of delay in the context of a reference clock

NTP supports attaching *reference clocks* to a computer. A reference clock is a device, such as a WWVB receiver, GPS receiver, or atomic clock, that provides a source of time to the computer. The NTP implementation I am using models such a clock as a network peer. For every measurement made to a reference clock, the delay is considered zero. However, there is an “error” value associate with a measurement. In the case of the WWVB clock and this implementation, the error value is larger when the clock indicates that its accuracy is lower.

Thus, for a reference clock, I use the error value as the delay, and process the observations according to the above algorithm. This is intuitively pleasing, since the computation of predicted time uses delay as an estimator of error. Figure 3.8 shows an example of error indications from a reference clock.

```
LOW DELAY  64.438 ms  HIGH DELAY 151.184 ms
1%      5%      10%     25%     50%     75%     85%     90%     95%     99%
66.96   67.92   68.86   70.80   73.82   78.43   83.50   87.94   98.83   116.67
p_base 67.917 p_add 73.822 p_mul 78.430 add 5.905 mul 0.154792
DELAY: ORIG 1622 OK 1126 INVALID 496 (0.306 invalidated)
```

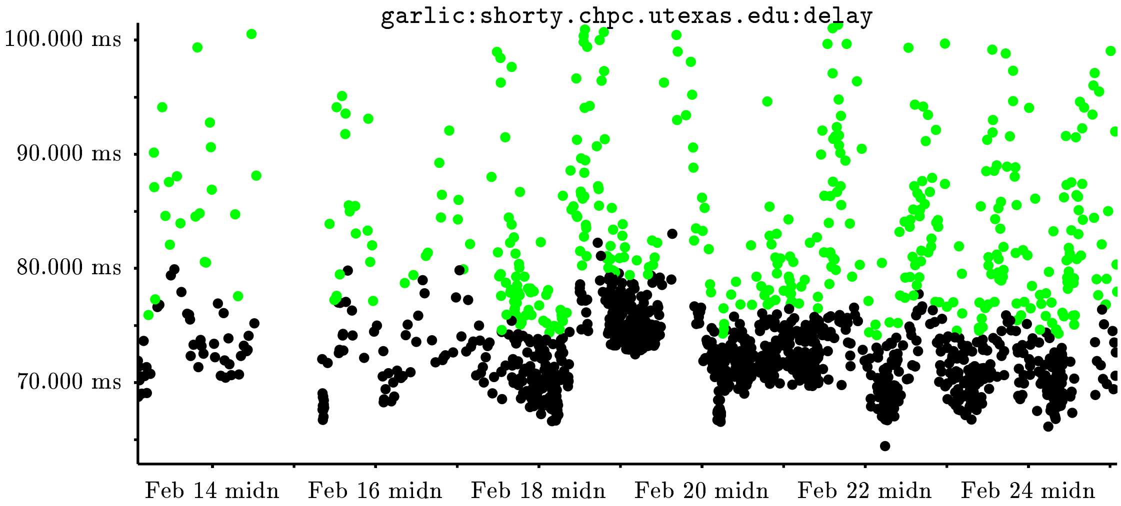



Figure 3.7: Statistical information and delay values for observations made from garlic to remote clock shorty.chpc.utexas.edu. A few very high-delay observations are omitted. The green points have been marked invalid because they are high delay.

### 3.4.3 Division into segments

The offset observations are divided into a number of segments. This is done for several reasons. One is to obtain an estimate of the average rms error caused by network delay variance. Another is to identify intervals in which the rms error over that interval is large relative to other intervals in order to discard points believed to be erroneous. The last is to provide a starting point for a process that repeatedly combines small segments to obtain intervals over which the local oscillator can be characterized according to the model. While one could argue that different methods of division are appropriate for these separate needs, I have found that the method presented here is satisfactory for all three uses. Later sections discuss the different needs and argue that the method presented in this section is satisfactory.

To perform the division, all of the observations are examined in turn (note that either only NTP data or data from one only remote clock are being examined). A new segment is begun if the current observation belongs to a different run than the current segment. Invalid observations are simply added to the current segment. A new segment is begun if the current segment contains the maximum number of points (128 in the "current version"). A new segment is also begun if the current segments contains at least a minimum number of points (16) and the addition of the current point would cause the difference of the time of the first and last point of the segment to be greater than a maximum time (4 hours). Thus, given the choices of constants indicated above, all segments (except possibly the last

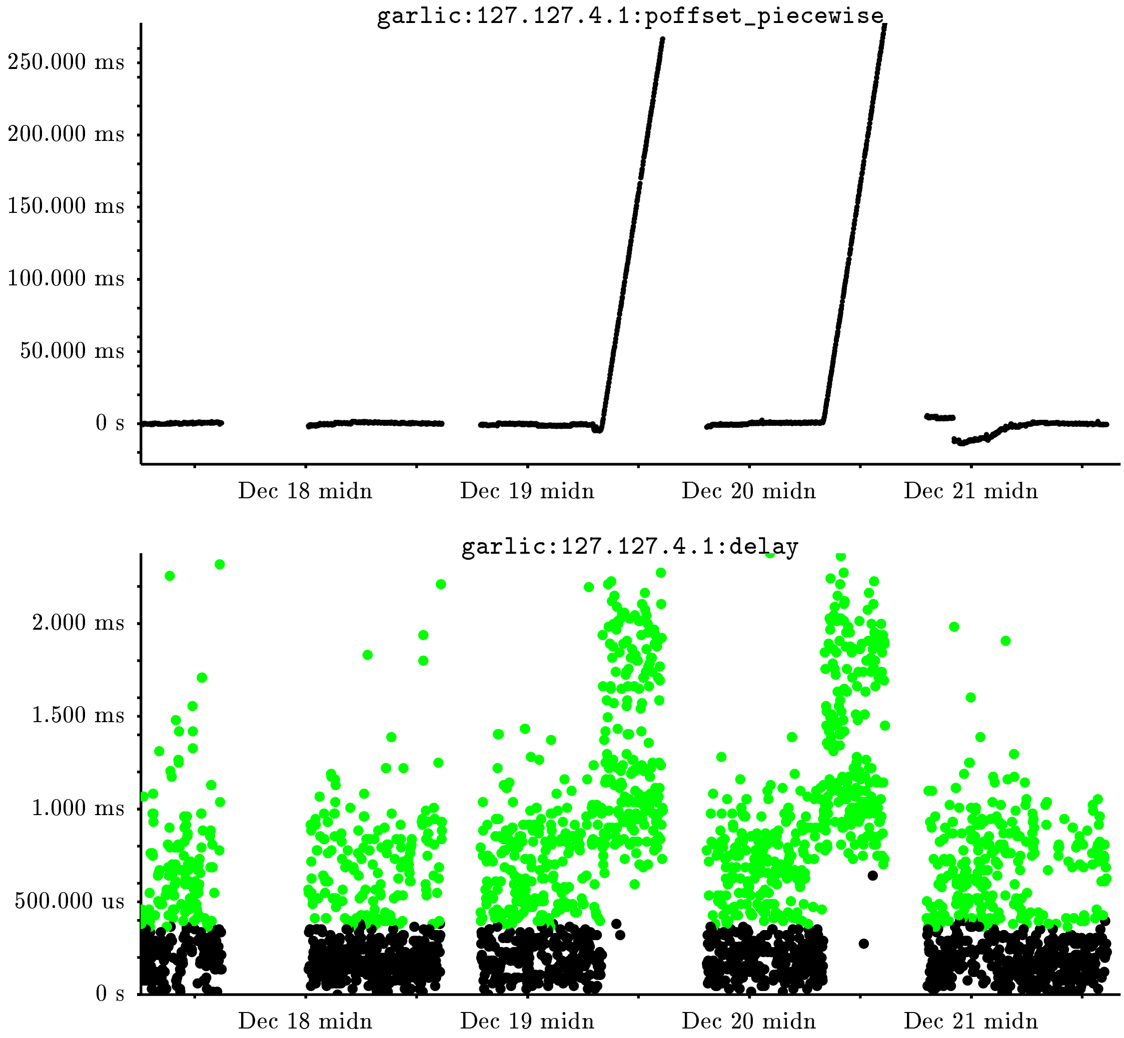



Figure 3.8: Predicted offsets, above, and error values (labeled delay), below, for a WWVB receiver attached to garlic, and statistical information about the error values. Note that erroneous values of `poffset` correspond to times when the error values are higher. Data are missing because at times the WWVB receiver did not output timing information.

of each run) contain points from only one run, at least a 16 valid points, at most 128 points, and contain points covering at most 4 hours.

An estimation of local oscillator parameters is then performed for each segment. Generally, the algorithm requires that estimated parameters, residuals and rms errors be available for a segment. Thus, the implementation of the algorithm generally estimates parameters and computes rms errors whenever a segment is created or modified. Throughout the rest of the description, I shall simply refer to such quantities; it should be understood that an implicit statement that they are calculated is being omitted to avoid clutter.

```
RMS ERRORS of 19 segments (LOW 0.000 ms, HIGH 0.336 ms):
1%      5%      10%     25%     50%     75%     85%     90%     95%     99%
0.00    0.00    0.23    0.25    0.26    0.29    0.32    0.32    0.34    0.34

RMS ERRORS of 18 segments (LOW 0.684 ms, HIGH 6.122 ms):
1%      5%      10%     25%     50%     75%     85%     90%     95%     99%
0.68    0.68    0.85    1.03    1.33    3.28    4.53    4.83    6.12    6.12
```

Figure 3.9: Sample output from median error calculations for two remote clocks.

### 3.4.4 Calculation of median error and invalidation of high-error segments

The median value of the rms error for the segments is calculated. All valid points belonging to a segment with an rms error at least twice the median rms error are marked invalid. The motivation for this is to mark as invalid groups of nearby observations that are not self-consistent on the theory that such observations are likely to be incorrect. As with a notion of "high delay," I must base the notion of "inconsistent" on information derived from the data itself. In contrast to the low-delay selection scheme, the notion of inconsistency is more complex — delay is a property of a single observation, while inconsistency is a property of a group of observations. Thus, the computation determines the median level of consistency of groups of points, rather than the median delay.

The question naturally arises as to how large these groups should be. The computation is attempting to find observations that are inconsistent over the short term due to errors at the remote clock, and is not attempting to determine the frequency stability of the local oscillator. Thus, the groups should be large enough to compute a meaningful rms error, but not so large as to cover time periods large enough to allow errors due to frequency changes of the local oscillator to become large. Thus, the length of segments was limited to at most 4 hours. This is intended to be small enough to keep errors from the type of frequency changes commonly observed for poor-quality oscillators fairly small. It is intended to be large enough to cause averaging over enough samples to reduce the effects of network noise, and to be able to fairly characterize the stability of the remote clock. If only a few samples were considered, one might simply estimate the effective frequency of the remote clock, rather than observe that its effective frequency changed.

In many graphs in this document, observations which have been marked invalid because they are part of a segment with a high rms error will be red.

Figure 3.9 shows the output of the calculation of median error for the same time period for two different remote clocks. The median error and various other order statistics are given. The first case is typical of a well-behaved remote clock and a reasonably good local oscillator; the maximum rms error is not grossly greater than the median. The second case is indicative of a remote clock that at times exhibits unstable time, or perhaps a local oscillator with similar instabilities; the maximum and 90th percentile rms error are both much greater than the median error.

The rationale for invalidating segments with relatively large rms errors is that such segments are symptomatic of underlying error sources, such as particularly high or strange network delay variance for a period of time, or incorrect time on the remote clock. In

Chapter 4, I will present examples of such behavior.

### 3.4.5 Invalidation of high-residual points vs. fits for segments

For each segment, all points with a residual greater than twice the rms error for the segment are marked invalid. The intent is to remove from consideration points that are "outlyers" with respect to their neighbors. Typically only a few percent of the points that are valid before this step are marked invalid by it. Observations which have been marked as invalid in this step will be shown as blue.

### 3.4.6 Procedure for computing "measure of similarity"

In the summary of the computation of piecewise predicted time, I stated that adjacent segments are examined and combined if the measure of fit describing how well they can be described by a single set of parameters was sufficiently high. I describe here how to compute a measure of similarity in several ways, discuss advantages and disadvantages of each, and explain the chosen scheme.

The procedure for computing measure of similarity takes as input two segments, that is, two periods of time and associated observations. The output is a single value indicating how well the observations contained in these two segments may be characterized by a single set of parameters; a lower value indicates a better characterization. I desire that the output value have roughly the same sense and meaning as the values rms error discussed earlier. That is, the measure of similarity of two segments should be roughly the same as the rms error of the result of combining the two segments unless the two segments are significantly different in character.

Earlier, I have referred to the rms error of a segment as a way of indicating the quality of the estimation. A straightforward way of computing measure of similarity is to produce estimates of phase(s), frequency error, and aging for the aggregation of observations of both input segments, and to compute the rms error of the aggregation of both sets with respect to these estimates. This "joint rms error method" is intuitively pleasing, as it essentially builds on the central notion of least-squares estimation that the best estimate is the one that results in the smallest rms error. In this case, I consider data to be better described by an estimate if the rms error is low.

The "joint rms error method" has a serious problem. I am concerned not only with how well a set of estimates describes a set of data *on average*, but with how well two sets of data *may be characterized* by a single set of estimates. The crucial difference is that a single set of estimates may have a small rms error with respect to the aggregation of two sets of observations, but in fact describe one of the sets poorly. Consider the case of two sets of observations. Assume that the first set has 1000 observations, the second 10, and that the rms error of each set with respect to the estimates describing it is 1 ms. Assume also that the frequency estimates for the two sets are significantly different. The estimated parameters for the joint set will be very close to those of the first set, rather than the second, since the estimation will be dominated by the 1000 points. The overall rms error for the joint set will be slightly higher than the error for the original sets. One can view part of the increase as coming from the fact that for the 1000 points of the first set the new estimated parameters are slightly worse than the set for just the 1000 points, and part as coming from the fact that the new estimates do not describe the second set very well. While the "joint rms error method" would give a low value in this case, the data do not fit together well at all.

Consider another method intended to rectify this problem, which I term the "rms error from other estimate method." Rather than calculating estimates to characterize the combined data, it is determined how well the estimates describing each set of observations characterize the other set. That is, residuals of the observations of each set with respect to the estimated parameters from the other set are computed. Then, the rms error of these two sets of residuals is computed. The higher of these two errors is used as the measure of similarity. This is intuitively pleasing as the method declares two sets of data to be characterizable as one if the estimates from each set describe the other well. Considering the previous example of two sets of observations with 1000 and 10 points, one would presume that neither rms error calculated would be small.

The "rms error from other estimate method" has a serious problem — it is in effect depending on extrapolation. Let us consider the previous example, except this time consider that the data are in fact characterizable by the same estimates, and that the estimates of the two sets are similar but not identical, due to noise in the data. The frequency estimate and resulting `poffset` values from the 1000 point set are likely to be reasonably accurate when applied to the 10 point set. One reason is that the estimates from the 1000 point set are likely to have a smaller amount of noise due to the greater number of samples. Another is that the time extent of the 10 point set is likely to be small compared to the time extent of the 1000 point set, and the application of the 1000-point estimates to the 10 point set involves extrapolation over only a small fraction of the time of the base data. The other rms error, that of the 1000 point set with respect to the estimates of the 10 point set, however, is not so likely to be small. If points are equally spaced, computing this rms error will in effect be extrapolating for a time period a factor of 100 greater than the extent of the base data. A small difference between the frequency error estimate and the true frequency error will have a much greater effect over this time period. Thus, this scheme yields unsatisfactory answers.

In order to rectify the deficiencies of the above schemes, I devised a more complex scheme with elements from each of the above schemes. Let $O_1$ be the first set of observations, and $E_1$ the estimated parameters for the first set, and similarly for the second set. Let $O_j$ be the joint set, and $E_j$ be the estimated parameters for the joint set. Let $R(O, E)$ be the rms error of $O$ with respect to $E$, and let us abbreviate $R(O_a, E_b)$ as $R_{ab}$. In these terms, the first scheme computes $R_{jj}$. The second scheme computes $\max(R_{12}, R_{21})$. The new scheme first calculates $R_{jj}$, which is exactly the value determined by the "joint rms error method." This is used as a lower bound on the value chosen by the new scheme; the measure of similarity is at least as bad as the value returned by the "joint" method. The new scheme then calculates $\max(R_{1j}, R_{2j})$, which is the greater (i.e., worse) of the measures of how well each original data set is described by the joint estimates. This is similar to the result of the "rms error from other estimate method," and one could consider returning as the result the larger of this value and the "joint" error $R_{jj}$.

While such a scheme appears pleasing, it has a subtle problem. Consider $O_1$ and $O_2$ such that each is reasonably characterizable by the same estimates, but let $O_2$ have a high rms error, i.e. let $R_{22}$ be large. In this case, $R_{2j}$ may be only slightly greater than $R_{22}$. In this case, $E_j$ really does describe $O_2$ reasonably well — only slightly worse than $E_2$. Thus, in order to separate the two separate causes of a large $R_{2j}$ — that $R_{22}$ itself is large, and that $E_j$ does not describe $O_2$ — the scheme calculates the ratio of how well each data set is described by the joint estimates compared to how well it is described by its own estimates: $R_{2j}/R_{22}$. In cases where this ratio is large, I wish to accept the high value of $R_{2j}$ as correctly indicative of a poor match. In cases where this ratio is small, such as the example, I wish to somehow discount the high value of $R_{2j}$. The scheme calculates the value of the joint error worsened by the above ratio, intended to capture the notion that the joint error is approximately what is desired, except that a higher value is desired if the joint estimate $E_j$ describes one of the sets much worse than its own estimate. For the first data set, this "worsened joint error" is

$$W_1 = \frac{R_{1j}}{R_{11}} R_{jj}$$

Now that the required intermediate quantities have been specified, I can specify the ultimate result of the computation. The result should be at least $R_{jj}$, the joint error. The result should be at most $\max(R_{1j}, R_{2j})$, the maximum of the errors of each data set with respect to the joint estimate. Also, the result should be at most $\max(W_1, W_2)$, the larger of the worsened joint error derived from each data set. Thus, the desired result is

$$\max\Big(R_{jj}, \min\big(\max(W_1, W_2), \max(R_{1j}, R_{2j})\big)\Big)$$

### 3.4.7 Combining segments that "fit together"

In the brief description of the computation of predicted time, I stated that adjacent segments are combined if the measure of similarity is sufficiently good, and I have now defined a procedure for computing the measure of similarity. I also introduced a notion that the intervals produced by the computation should be as large as possible, subject to the constraint that the offset observations be "reasonably described" by the estimates, and went on to adopt a notion that this meant that the rms errors of the data with respect to the estimates due to network-induced noise should be comparable to the errors due to unmodeled behavior of the local oscillator.

I discussed how to determine the precise meaning of "reasonably described," and now describe a procedure for repeatedly combining segments. The essence of the procedure is to repeatedly combine the adjacent segments for which the measure of similarity is smallest (i.e., best), continuing until no measure of similarity is smaller than a value termed the "combining limit." The combining limit attempts to capture the notion of "reasonably described." Also, a procedure for deciding to move observations between adjacent segments is described. Essentially, the procedure attempts to change segment boundaries slightly if doing so results in lower rms errors. This "migration" procedure and the reason it is needed are discussed in future section.

The incremental combining and migration procedure may appear ad-hoc or unclean. Consider instead a "clean" procedure to specify the intervals to be produced by the piecewise computation. Consider the set $S$ of all sets of intervals. Consider a member of this set $S_i$. Let the member $S_i$ be considered *acceptable* if the largest value of rms error of the intervals of $S_i$ is less than or equal to the combining limit. Let $S_a$ be the subset of *acceptable* members of $S$. Let $S_{min}$ be a member of $S_a$ with minimum size, i.e., a set of intervals such that there is no other member with fewer intervals. Let $S_b$ be the subset of members of $S_a$ with the same number of intervals as $S_{min}$, or the set of *acceptable* members with the minimum number of intervals. Now, choose $S_{result}$ as the member of $S_b$ that has the lowest value over all its member intervals of maximum rms error, and let this set of intervals and associated estimates be the result of the computation.

While the above scheme is intellectually pleasing — it selects the set of intervals of minimum size and the smallest maximum error — it is unacceptably time-consuming to implement. The procedure actually used is intended to produce a similar result. It produces a result more consistent with the real goals of the computation — identification of intervals over which the local oscillator behavior may be reasonably characterized — because it combines segments that are more similar first, rather than finding a set of intervals with a minimum maximum error. The formal description is given to illustrate the complexity of computing an exact answer in order to explain the use of a heuristic algorithm that computes a useful approximation reasonably efficiently.

### Determination of combining limit

Earlier in the computation, the data were divided into segments and the median of the rms errors of the segments was calculated. Let us make several assumptions, realizing that the assumptions are imperfect. Assume that the local oscillator behaved according to the estimates made over those short segments. Assume that observations of the remote clock during times the remote clock exhibited anomalous behavior have been marked invalid, in accordance with the intent of the process of marking as invalid segments with high rms errors. Given these assumptions, I may view the rms error of the remaining segments as being caused solely by network delay variance. Then, I may interpret the median rms error as a single-number characterization of the level of rms error due to network delay variance. I choose the median because it is not affected by the tails of the distribution, and as a hedge against the fact that the above assumptions are not completely true.

Given a value of "rms error caused by network delay variance," or $E_n$, I can now consider choosing a value for the combining limit. I stated that I wished the errors due to network delay variance to be comparable to those from other sources, so let us consider choosing a value of $2E_n$ as a combining limit. While intuitively pleasing, this value is not fully satisfactory. Recall that all observations in segments with an rms error greater than $2E_n$ were marked invalid. This implies that there may be segments ready for combining with an rms error of just less than $2E_n$. Consider the possibility of two segments adjacent to each other, each with an rms error of $1.9E_n$, and with a measure of similarity of $2.1E_n$. I would in fact like these segments to be combined; the "additional error" is only $0.2E_n$, but the measure of similarity is greater than the combining limit. Thus, for the computation of piecewise predicted time for analysis purposes, I choose a limit of $4E_n$, twice the possible rms error of a remaining segment. This, however, may result in error contributions from unmodeled local oscillator behavior of roughly $3E_n$.

Based on examining a large amount of actual data, it appears that the value of $4E_n$ is reasonable for analysis purposes; Figure 3.10 shows frequency estimates and predicted offsets resulting from this choice of combining limit. It is not too small; a much smaller

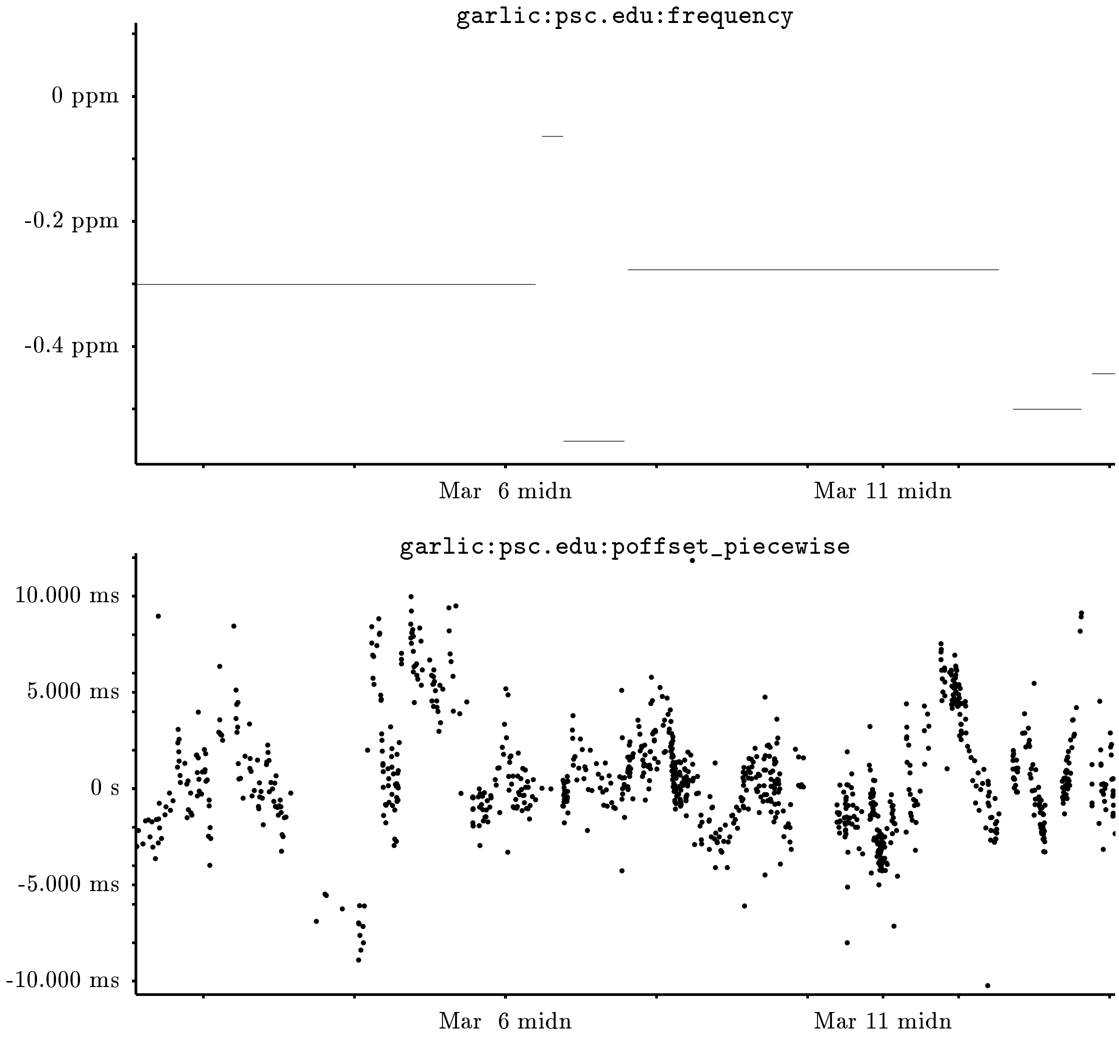



Figure 3.10: Frequency error estimates (in ppm), above, and predicted offsets of psc.edu from garlic, below, with a combining limit of $4E_n$.

value would cause segments not to be combined due to locally high network delay variance. It is not excessively large; a much larger value would cause segments to be combined when significant frequency changes in the local oscillator had occurred. For analysis, there are many possible choices of a combining limit which are each reasonable but will produce different results. A different limit should be chosen depending on the intent. If one is interested in understanding the frequency changes of the local oscillator, a value of $2E_n$ or $2.5E_n$ may be appropriate; Figure 3.11 shows results with such a combining limit. With this value, some segments which perhaps "should have been" will not be combined. If one is interested in a broader view of the data, and wishes to see the effects of minor frequency changes in the context of the network noise, a larger value such as $4E_n$ is more appropriate. In this case, the intervals are longer, and one can see the effects of the unmodeled frequency errors on the values of `poffset`, rather than having separate frequency estimates. In the

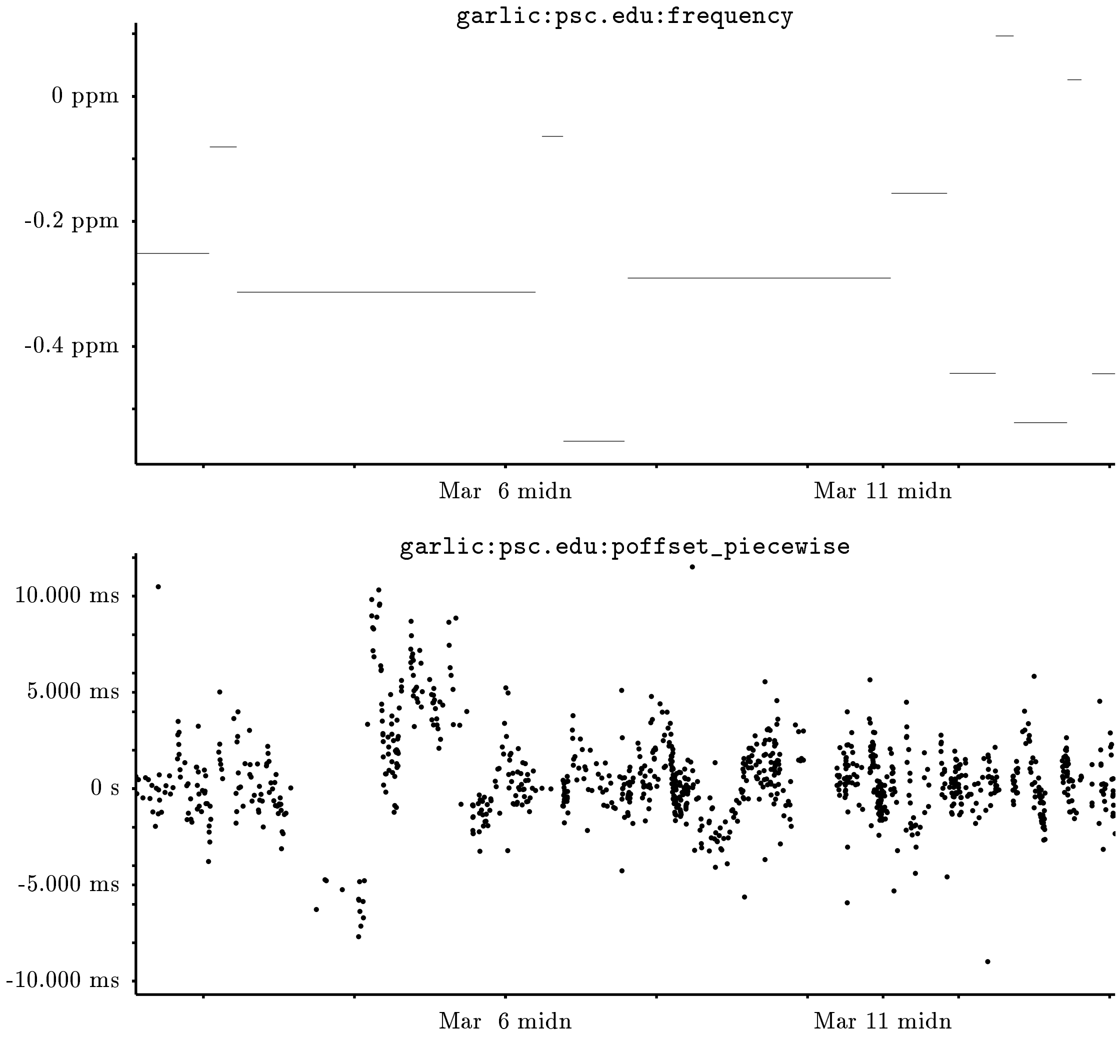



Figure 3.11: Frequency error estimates (in ppm) and predicted offsets of 128.182.62.100 from garlic with a combining limit of $2.5E_n$.

extreme case, a combining limit of $\infty$ produces results similar to the overall computation, except that some offset observations are marked invalid.

### Migration of observations across segment boundaries

Conceptually, I would like to start the combining process with segments consisting of only one observation; beginning as the computation does with small segments seems unreasonable as there is no reason to believe the observations in a given segment may be reasonably described by a single set of estimates. However, with only a single offset observation, frequency may not be estimated, and no meaningful value of rms error is possible. The concepts on which combining is based require the segments to have a sufficient number of points to compute estimates of local oscillator behavior as well as meaningful values of rms errors.

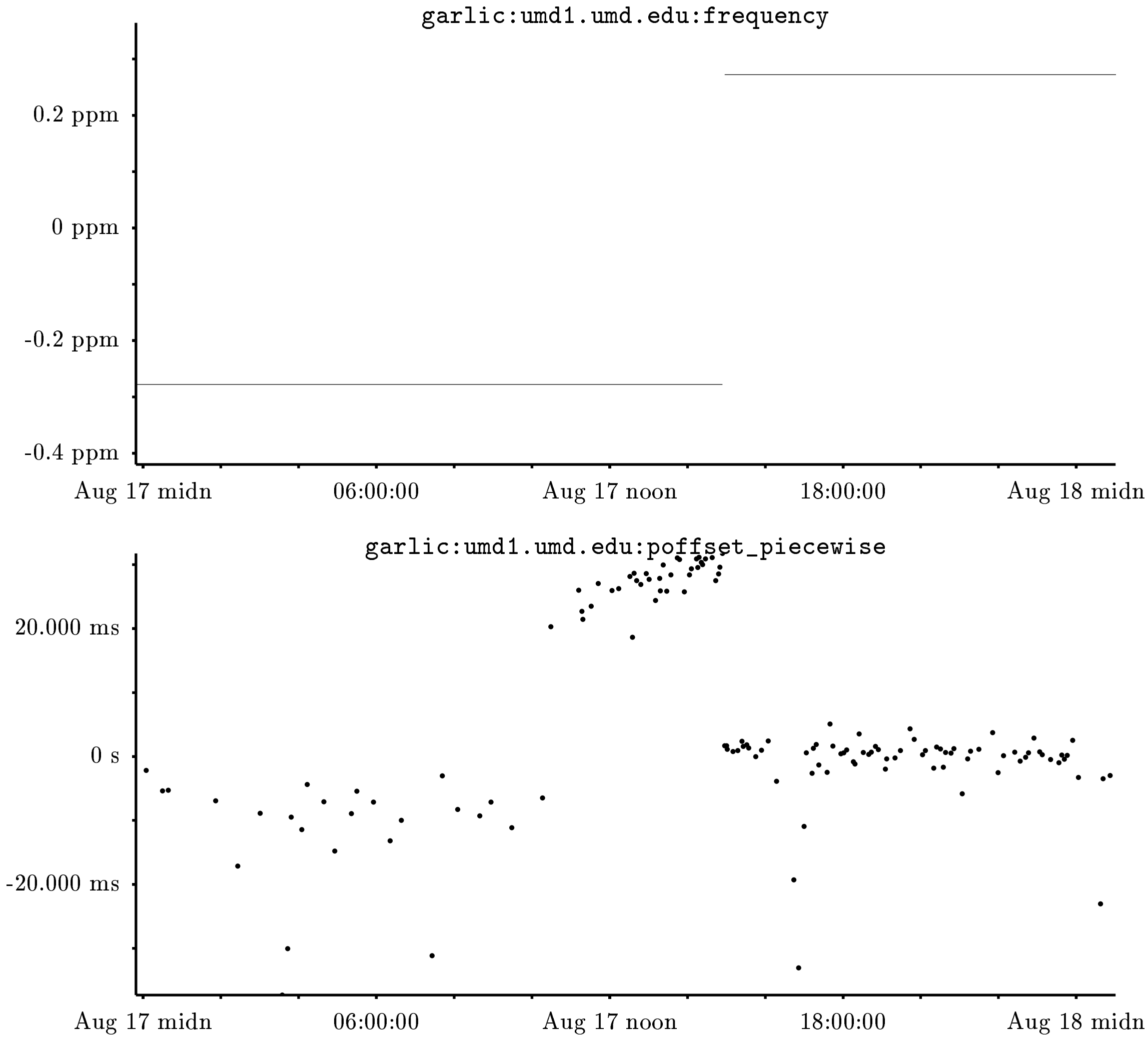



Figure 3.12: Frequency estimates and predicted offsets near a change in frequency of the local oscillator. The apparent discontinuity in values of `poffset` near 10 a.m. August 17 corresponds to the time the local oscillator was replaced.

For any given segment, one could perhaps argue that the observations comprising it would have been combined because the rms error of the segment is less than the combining limit. However, consider an example in which the local oscillator exhibited a small change in frequency error at a particular time which happens to be in the middle of a segment. Assume that the rms error of the segment containing the change is modest, perhaps $1.6E_n$. Assume that the local oscillator, remote clock and network were otherwise well-behaved, so that all segments before and after the frequency error change are combined. Assume that the segment containing the change will be first combined with the segment preceding it, and the end result of the combining process will be two segments. While this result is not grossly incorrect, some observations are contained in the "wrong" segment, according to our construction; Figure 3.12 shows such a situation. Consider whether the observations

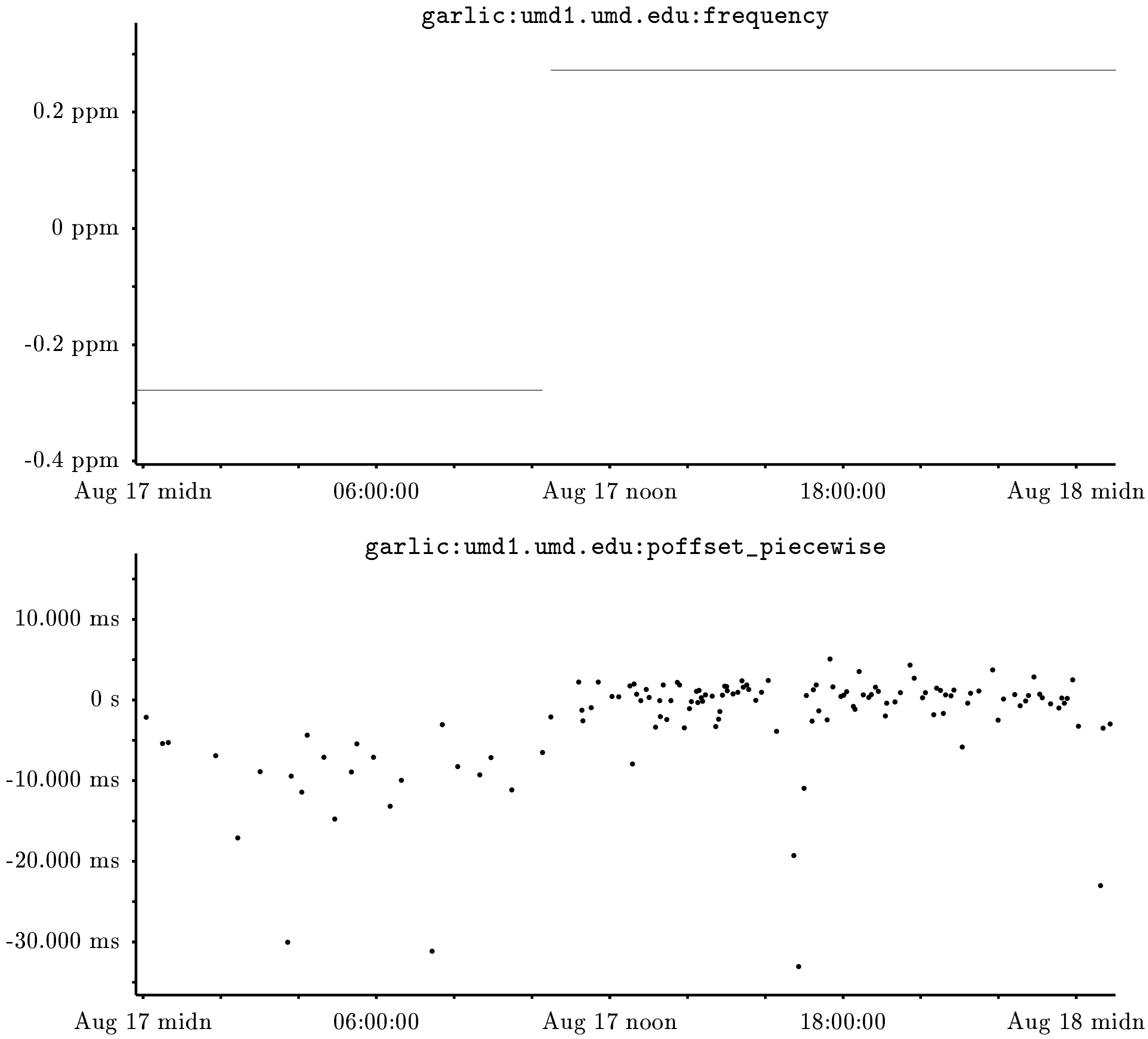



Figure 3.13: Frequency estimates and predicted offsets near a change in frequency of the local oscillator, with migration of points across segment boundaries.

are in fact in the wrong segment in some objective manner, rather than merely because I said they were. Observing the values of `poffset` in the figure, one can see that they are sharply higher in magnitude near the change of frequency. However, the values of `poffset` on the other side of the segment boundary are normal.

Thus, I introduce a scheme of *migrating* observations across segment boundaries if the residual of the observation with respect to the new candidate segment is significantly smaller than the value of the current segment. Precisely, every time a "measure of fit" is computed for two segments, and they are not combined, some points are possibly migrated across the segment boundary. Migration is only done if the segment to which points would be migrated already has a sufficient number of points (64, or 8 if it is larger than the other segment). First, the last point of the earlier segment is checked. If the residual of that point with respect to the estimates of the later segment is more than 5% lower than the residual of that point with respect to the estimates of its own segment, that point is selected for migration, and the next point is considered. At most 4 points are selected in this manner, and moved to the other segment. If no points were selected, the first points of the later segment are similarly checked. If any points are moved, the estimates are recomputed; the reason at most 4 points are moved is to move only a small number of points relative to the total so that the estimates remain close to the correct value while further points are being considered. After recomputing estimates, further migration is not immediately considered. Another attempt at migration is made following the next unsuccessful attempt to join the segments. Figure 3.13 shows the resulting values of `poffset` when the migration process was added; in this case the time of the frequency change was accurately identified.

### 3.4.8 Invalidation of high-residual points vs. resulting fits

For each segment remaining on the list, all points with a residual greater than twice the rms error for the segment are marked invalid. Again, the intent is to remove from consideration points that are "outlyers" with respect to their neighbors. As in the previous step where high-residual points were marked invalid, typically only a few percent of the points that are valid before this step are marked invalid by it. The process of marking invalid high-residual points is repeated at this time in order to discard points that were not "outlyers" with respect to the short segments, but are now outlyers with respect to the larger segments resulting from the combining process. Observations which have been marked as invalid in this step will be shown as yellow.

### 3.4.9 Estimation of missing parameters

For each segment, the estimates are examined. If an estimate does not contain a phase value for a given run, the phase for that run *given the frequency error and aging estimates* is estimated from all data for that run. The motivation for this step is that even if all observations for a particular run have been marked invalid and thus no phase could be estimated for that run, I would still like to compute meaningful values of `poffset` and `rsadjresid` for those observations in order to analyze behavior during that run. Thus, the computation estimates a phase for the run from all data, since there is no valid data.

## 3.5 Integrated predicted time

The computation of integrated predicted time attempts to produce a value of predicted time that takes into account raw observations from many remote clocks. In this way it is like the value of predicted time computed by NTP's system offset, except that the combining of remote clocks is done at the end, rather than before the predicted time computation. Thus, the problems of using NTP's system offset, such as the lack of ability to use delay information, and, most importantly, the effects of changes in clock selection, are avoided.

Integrated predicted time is useful for analysis, in that the behavior of a remote clock can be examined with respect to predicted time computed from multiple clocks. From this one can determine relative apparent offsets between remote clocks. By examining multiple remote clocks with respect to the same value of predicted time, one can determine whether lack of stability of the local oscillator or incorrect time from the remote clock caused high residuals for that clock. However, the most important use for integrated predicted time, and the motivation for its development, is as a synchronization algorithm. Because of this, the computation of integrated predicted time attempts to accurately characterize the local oscillator in the region of time immediately prior to the current time, i.e., the highest value of time in the input data. One could compute an "overall-integrated" predicted time that uses all observations from all clocks equally, but I have not found it helpful to do so.

The computation of integrated predicted time assumes that piecewise predicted time has already been computed for each remote clock. Points marked as invalid during the piecewise computation are considered invalid for the integrated computation. The computation of predicted time computes an estimate of local oscillator parameters given the valid observations contained in the most recent interval (of the piecewise computation) for most remote clocks. In addition to estimating a phase for each run, a frequency error and possibly an aging rate, a *systematic offset* is also estimated for each remote clock.

### 3.5.1 Estimating systematic offsets between remote clocks

Earlier I argued that it was not possible to differentiate between a systematic offset error on a remote clock and persistent asymmetric delays (Section 1.4.3, page 22). Given that remote clocks may well have systematic offset errors due to the manner in which they are synchronized to true time, and that asymmetric network delays are possible, it is likely that various remote clocks will exhibit different systematic offsets. Thus, I estimate such a parameter for each remote clock.

Let $o = (t, u, c)$ be an offset observation — an ordered triple of time, `uoffset`, and remote clock identifier — that is, an example member of the set $O$ of valid offset observations of the various remote clocks. Let $r(o)$ be the run identifier associate with $o$, and $t(o)$, $u(o)$, and $c(o)$ be the observation time, value of `uoffset` and remote clock identifier of $o$. I then compute estimates of $\hat{p}_r$, phases for each run, $\hat{s}_c$, systematic offsets for each clock, $\hat{f}$, frequency error, and $\hat{a}$, aging rate that result in the minimum rms value of `poffset`:

$$\sum_{o \in O} \left( u(o) - \left( \hat{p}_{r(o)} + \hat{s}_{c(o)} + \hat{f}(t(o) - t_0) + \frac{\hat{a}}{2}(t(o) - t_0)^2 \right) \right)^2$$

The estimates described above are not unique — adding a certain amount to each $\hat{p}_i$ and subtracting it from each $\hat{s}_i$ results in the same residuals. I choose estimates in such a way that the values of $\hat{s}_i$ are small.

### 3.5.2 Selection of remote clocks for inclusion

Earlier I indicated that offset observations from "most" remote clocks are included in the computation of integrated predicted time. Because the intent of integrated predicted time is to provide an accurate characterization of the local oscillator for the present and recent past, remote clocks are only included if they meet certain criteria. One of the criteria requires a notion of the "current time." I define the current time to be the largest of the time values of all offset observations, both NTP and raw. A remote clock is *not* included in integrated time if

1. it contains too few observations (in the current computation, fewer than 16), or

2. it spans too small a time (less than 4 hours), or
3. it is old, that is, if its most recent observation is too distant from the current time (more than 4 hours), or
4. the user has specified a candidate list of acceptable remote clocks and the clock is not on that list.

### 3.5.3 Non-uniform weighting of remote clocks

A weight is computed for each remote clock, and each observation of that remote clock is weighted when estimating parameters. The assigned weight attempts to favor remote clocks which are self-consistent.

Initially, each remote clock is assigned a value of 1. When offset observations are marked invalid because they belong to a high-rms-error segment, because they have a high residual with respect to their segment before combining, or because they have a high residual after combining, the value of weight is set to its old value times a function of the fraction of offset observations that are not marked invalid. If a fraction $i$ of the observations are marked invalid, the value of weight would be multiplied by

$$\frac{1}{\frac{i}{0.25}^{3}}$$

This function was chosen to impose a small penalty for a small fraction of invalidated observations, to impose a penalty of 50% for the case when one-fourth of the observations are marked invalid, and to impose a large penalty when more observations than one-fourth are marked invalid.

The method of assigning weights is perhaps overly simplistic. It is likely that careful study could produce a better scheme. Such an undertaking is beyond the scope of this thesis.

## 3.6 Deciding whether to estimate aging

Earlier, I stated that while phase and frequency error are always estimated, aging rate is only sometimes estimated. For some local oscillators and observed offsets, the true value of aging is significant and should be estimated, as in the figures on pages 34 and 35, where estimating aging rather than assuming it to be zero resulted in sharply lower residuals. For others the true value of aging is close to zero relative to the error in the aging estimate; many high-stability oscillators have low aging rates. In these cases, estimating aging is not harmful for analysis purposes, as the resulting residuals are little changed. However, when extrapolating from the frequency and aging estimates in order to synchronize the local clock, errors in estimating the aging rate can be more significant.

Given Gaussian error distributions, procedures exist for calculating the expected errors in estimated parameters ([BHB62], pp. 482–485). However, I have no reason to believe that the error sources are strictly Gaussian. Small shifts in systematic offset over a period of days can easily be misperceived as aging. If this were the case, I would be estimating a parameter based on systematic errors having nothing to do with the value of the parameter.

In some cases it is beneficial — almost necessary — to estimate aging, and in some cases it is harmful. I wish to avoid estimating aging if there is no benefit to doing so. The solution I have adopted is to have the user of the output of the computation of predicted time decide whether to estimate aging or not. Generally, aging would not be estimated unless observation of the output shows cause that it should be. To aid the user in this decision, the program implementing the computations of overall and piecewise predicted time optionally estimates parameters with aging and without aging, and reports the rms errors obtained with and without aging, and the ratio of the errors. In those cases where estimating aging is important, the rms error with aging will be significantly lower than the rms error with aging set to zero.

Even if the computation does estimate aging, the aging estimate is set to zero for sets of observations shorter than a configurable time period (48 hours). If the magnitude of the aging rate estimate is greater than a configurable amount (0.1 ppm/day), the aging rate estimate is set to zero instead. Similarly, if the magnitude of the frequency error estimate is greater than a configurable amount (1000 ppm), zero is used as the frequency error estimate. The motivation for the limits on parameters is that certain bounds are *a priori* known for the local oscillator; oscillators outside the bounds are considered broken. Thus, out-of-bounds values can be rejected. Rejection of excessive aging estimates causes the default value of zero to be used instead, so that aging is effectively not estimated. Rejection of excessive frequency error estimates causes a value of zero to be used instead. In this case, the rms error for the set of observations will likely be very high. This is appropriate because it has been the author's experience that an out-of-bounds frequency estimate is more likely to be caused by erroneous offset values or anomalous remote clock behavior than by a local oscillator truly having that large a frequency error.

## 3.7 Summary

I have described the computation of three types of predicted time. Overall predicted time is useful for the "big picture." Piecewise predicted time is useful for observing behavior in detail, determining which observations are erroneous, and identifying and examining changes in frequency error of the local oscillator. Integrated predicted time is useful for accurately characterizing the recent behavior of the local oscillator, and for estimating systematic offsets among the various remote clocks.

# Chapter 4
# Observations using the `rsadj` Technique

Using the `rsadj` synchronization measurement technique, I have made many observations about the behavior of local oscillators, the behavior of remote clocks, and the behavior of the packet networks used to observe the remote clocks. I have also made observations about the behavior of the NTP synchronization algorithms using the technique. With the insight gained from these observations, I suggest modifications to the NTP algorithms. I have verified the `rsadj` measurement technique by also observing synchronization with an external trusted reference — a GPS receiver with precise timing output.

By explaining the observations in this chapter, I hope to convey to the reader a sense of how to interpet the output of the `rsadj` measurement scheme. This is the most difficult issue in using the measurement scheme — deciding what errors or effects are due to what part of the system.

I present observations showing that some local oscillators exhibit frequency changes of several parts in $10^6$, and that others are stable to several parts in $10^8$. I show that the system clocks of some computers appear to malfunction. I show that some remote clocks behave as if the values of their system clock differ appreciably from true time. I show that one remote clock appears to wander back and forth, and that another is usually correct but exhibits repeated short periods of erroneous behavior. I show the effects of upgrading the hardware of a remote clock. I show the behavior of several typical well-behaved remote clocks. I present data showing the values of systematic offsets observed among various remote clocks, and that the differences in systematic offsets are far greater than the variations observed within a single remote clock.

I present observations of two radio timecode receivers configured as reference clocks. I show that one of these receivers outputs erroneous values of time. Before constructing the `rsadj` synchronization measurement technique, I was not able to tell that this receiver did not function well.

I present data showing the synchronization behavior obtained when running NTP — both the required parts of the protocol and the recommended algorithms. I show the undesirable synchronization behavior induced by large systematic offsets among remote clocks and short periods of incorrect observations. Based on the observations presented here and many more observations of NTP's behavior that are not presented, I give suggestions to improve the performance of NTP.

I present a method for verifying the performance of the `rsadj` synchronization measurement technique by using a GPS receiver as a source of true time. I explain how one can transfer GPS-derived time across an ethernet, and how one can compute a quantity showing the behavior of the local oscillator relative to GPS time. I present data showing that the basic assumption made by the `rsadj` technique, that the local oscillator can be characterized, is sound. I show that views of remote clocks with respect to the local oscillator obtained by the `rsadj` technique are very similar to views of remote clocks relative to GPS time.

## 4.1 Experimental apparatus

Several computers and oscillators were used to develop and verify the `rsadj` technique and the integrated timekeeping scheme. The arrangement described here may not seem entirely natural; it is influenced by several factors having to with the local physical plant and computing facilities. In particular, a computer being attached to the GPS receiver and accessible were mutually exclusive conditions.

### 4.1.1 Oscillator testbed — garlic

I installed a DEC KWV11-C real-time clock board in a DEC VAXstation 3200 named garlic. This board, as used in this work, essentially provides a 16-bit counter driven by an external oscillator. Various external oscillators, mostly at 5 and 10 MHz, were prescaled to 500 kHz — the maximum rate supported by the KWV11-C — and then connected. Oscillators connected included a Crystek oven-controlled crystal oscillator (OCXO), a Bliley OCXO, and an EG&G TCXO. The Crystek and EG&G oscillators cost roughly $100 each new in single quantity. The Bliley oscillator was obtained at an amateur radio flea market; an oscillator of comparable quality would cost roughly $500–$1000 new.

I modified the operating system kernel to use the KWV11-C counter rather than the normal system clock facilities. At each interrupt of the scheduling clock, the number of 2 $\mu$s ticks of the external oscillator since the last interrupt are read from the KWV11-C, and system's value of time increased appropriately. This process of reading the count of ticks and increasing the system clock value is also performed whenever the time is requested by a user process.

### 4.1.2 External trusted reference support — ipa

A custom-designed and built board was installed in a DEC DS5000/25 named ipa. This board contains a 10 MHz oscillator, a 64-bit counter driven by the oscillator, and a 32-bit capture register which records the lower 32 bits of the counter on the rising edge of an external signal. I connected the 1 pulse-per-second output of a Magellan GPS receiver to the capture register control input of the board. I installed a GPS antenna on the roof of the LCS building, and installed the computer and GPS receiver near the antenna on the highest floor. The computer was then connected to the same ethernet as garlic.

### 4.1.3 Other machines

I gathered data from two other DEC VAXstation 3200s without special hardware, named pepper and mace, located in offices. I also gathered data from a DEC MicroVax 3600 named larch, which was in a machine room. I also gathered data from two DECStation 5000/240s, named ginger and larch, which were located in machine rooms. The name larch is associated with two separate physical computers; the research group using it upgraded its computer hardware.

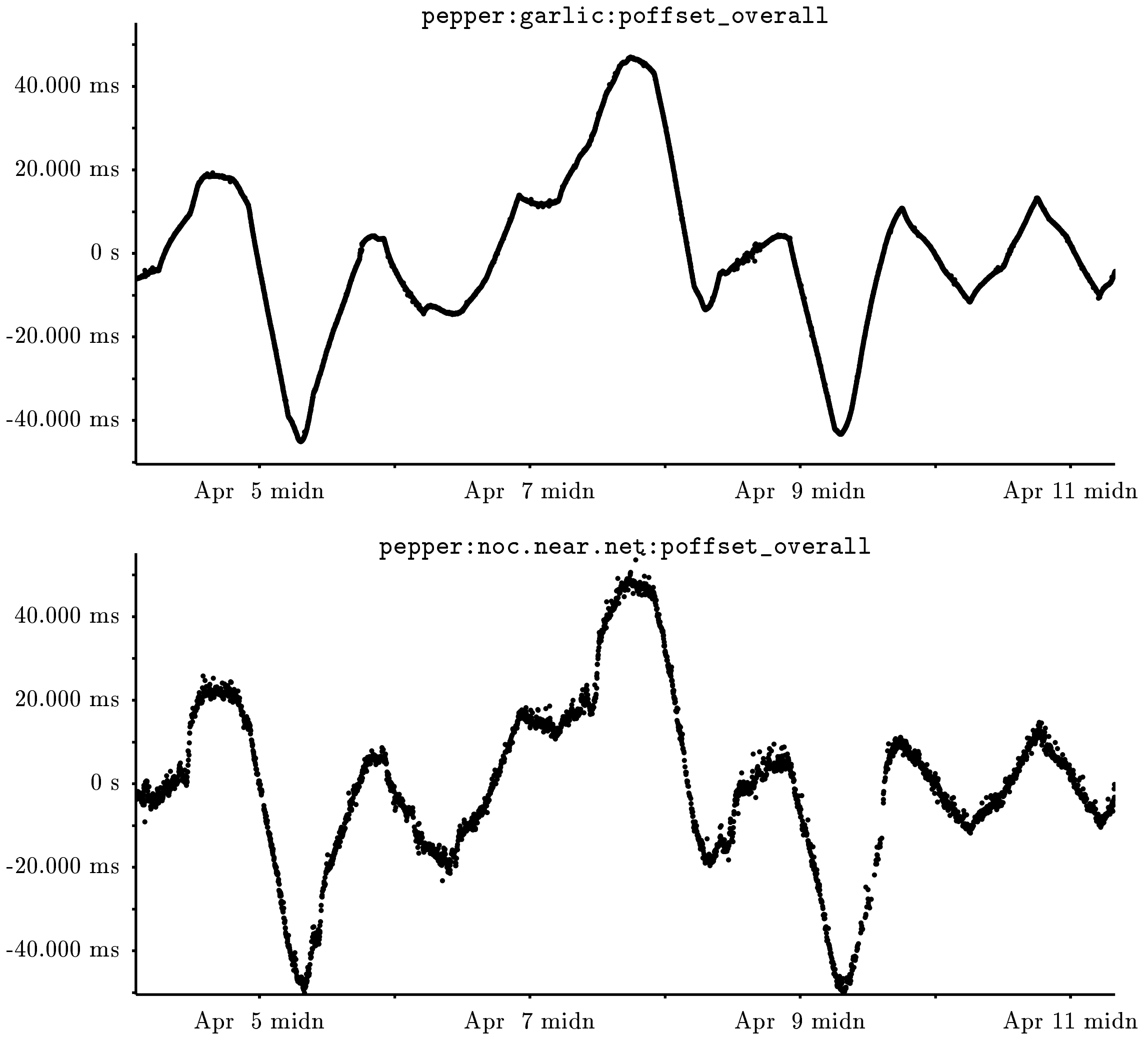



Figure 4.1: Overall predicted offsets of garlic, top, and noc.near.net, bottom, as observed from pepper.

## 4.2 Observations about local oscillators

From the data gathered, I was able to deduce information about the stability properties of various local oscillators. While observations of one remote clock with respect to one local oscillator are not usually sufficient to draw conclusions about either the local oscillator or the remote clock, observing multiple remote clocks often allows one to make inferences about both the local oscillator and the remote clocks.

### 4.2.1 Supplied local oscillators

I examined the local oscillators supplied with various computers. Generally, I found that they had fairly poor stability, changing in frequency by several ppm on a daily basis. It appears in many cases that the frequency changes are related to temperature changes. In the oscillators I observed, the day to day frequency changes were of sufficient magnitude that aging effects were not apparent.

Figure 4.1 shows overall predicted offsets of garlic and noc.near.net as observed from pepper. The data clearly show the same trends — the most obvious difference between the two graphs is that the variation from the trend line is greater for noc.near.net. This is to be expected, as it is several hops away on the network, while pepper and garlic are on the same ethernet. Given two such similar graphs, it is much more likely that the local oscillator of pepper changed frequency than that the two remote clocks had identical excursions in phase. Inspecting the inflection points, I can see that they occur approximately twice daily, once at roughly 6 a.m. and once between 6 p.m. and midnight — plausibly corresponding to changes in office temperature due to insolation and occupancy.

Figure 4.2 shows data covering exactly the same time period as Figure 4.1, but instead shows the output of the piecewise computation of predicted time. The values of `poffset` — the residuals of the estimation of oscillator parameters — are much smaller, by approximately a factor of ten. The middle graph is of the frequency estimates and intervals produced by the piecewise computation, and clearly shows the frequency changes that occurred. Note that the discussion of the quality of the local oscillator in pepper is decoupled from the actions of the synchronization algorithm running at the time. Values of `aoffset` for the same time period are small and roughly zero-mean, indicating that the standard NTP algorithms running during this time functioned well.

### 4.2.2 Medium-grade and high-grade oscillators

It is also interesting to examine the behavior of better local oscillators. Figure 4.3 shows predicted offsets of two remote clocks observed by garlic during a time when an EG&G TCXO, or temperature-compensated crystal oscillator, was the local oscillator of garlic. This oscillator cost roughly $100 in single quantity and has modest power supply requirements. The top graph shows overall predicted offsets of mizbeaver.udel.edu, a computer usually synchronized to a GPS timing receiver. The bottom graph shows overall predicted offsets of apple.com, a computer usually synchronized to WWV/WWVH (HF radio timing signals from NIST at Ft. Collins, CO, and Hawaii.).

One obvious feature of the two graphs is that the variance — presumably due to variance in network delay — of the offsets to mizbeaver.udel.edu is much less. A close comparison, however, shows very strong similarities in the trend lines of the two graphs. Examining the features of the upper graph, one can see them reflected — although less clearly — in the lower graph. Thus, I can conclude that the features are due to unmodeled local oscillator behavior, rather than identical incorrect remote clock behavior. One important feature of the lower graph is not reflected in the upper one — the group of observations with value roughly −8 ms near March 6. Because this feature appears in data from only one remote clock, there is no reason to attribute it to the local oscillator.

## 4.3 Observations about system clock quality

I have stated that the system clock counts ticks of the local oscillator, and have dealt previously with local oscillators of high accuracy relative to the observable error sources. Sometimes, however, the resolution of the local oscillator and system clock is quite limited.

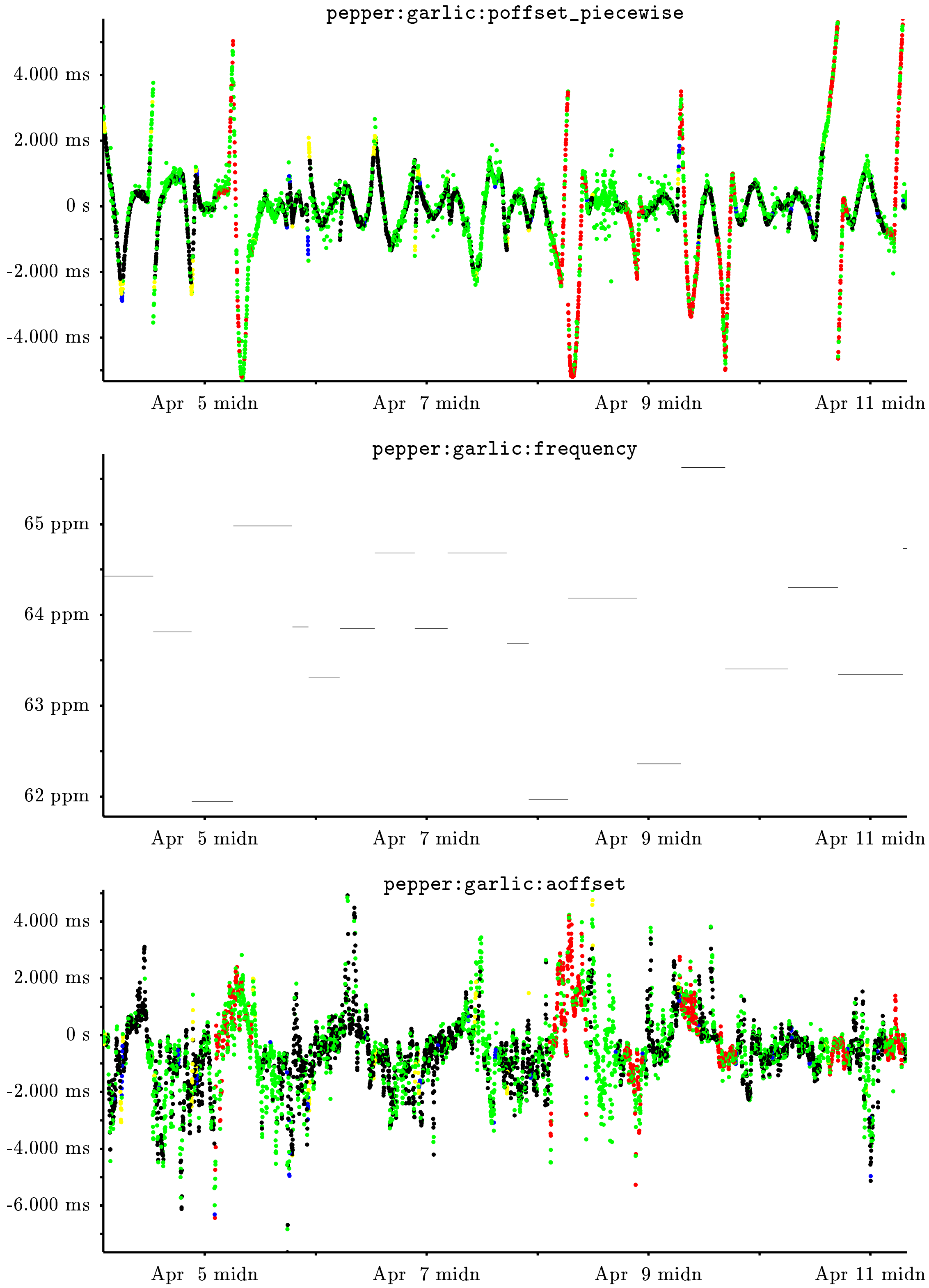



Figure 4.2: Piecewise predicted offsets of garlic, top, and the corresponding frequency estimates (in ppm), as observed by pepper, middle. Actual offsets of garlic as observed by pepper, bottom.

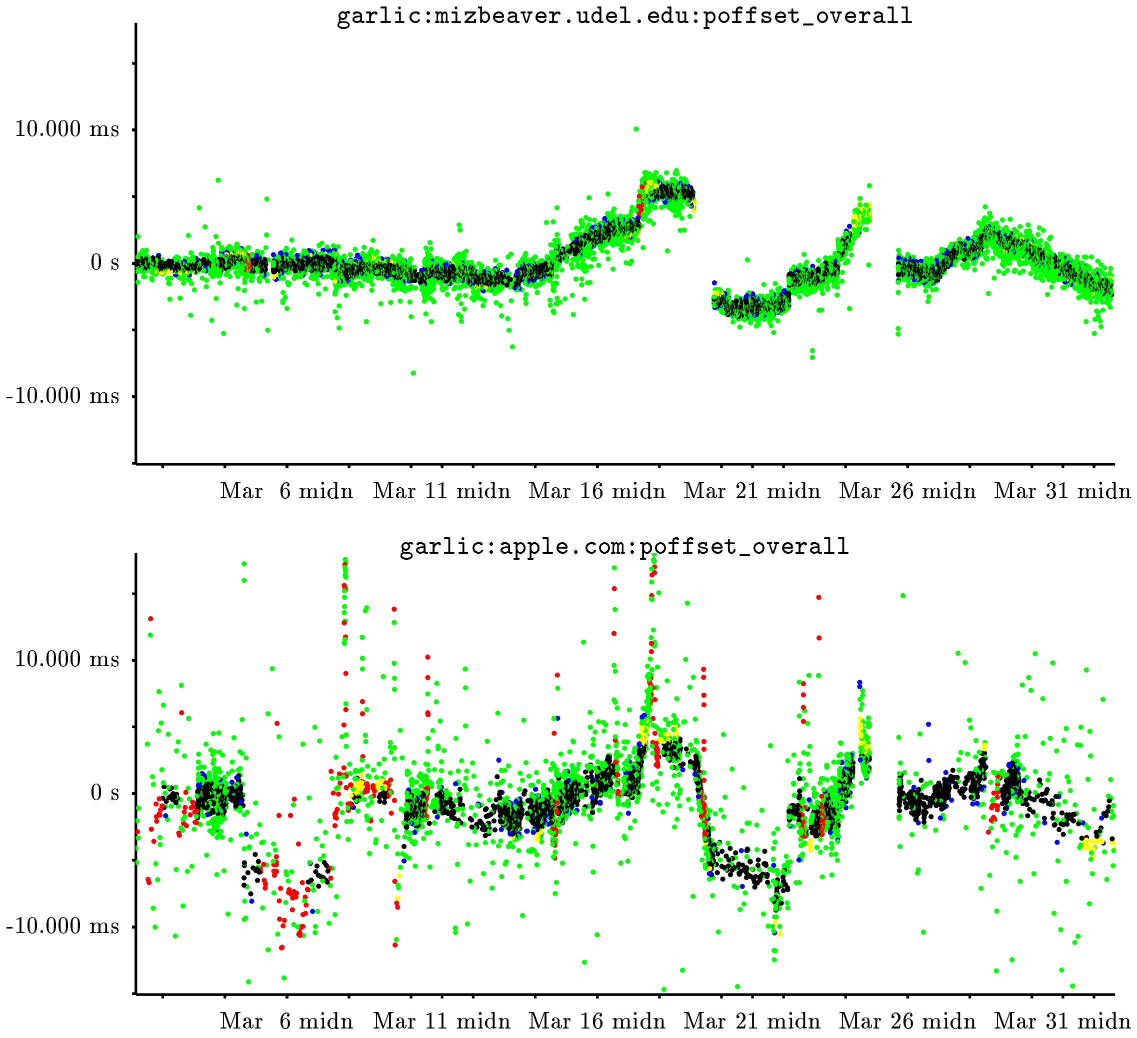



Figure 4.3: Overall predicted offsets of mizbeaver.udel.edu and apple.com as observed by garlic.

For example, DECStation 5000's generally have a clock interrupt rate of 256 Hz, and provide no timing facilities of greater resolution. Thus, the value of the system clock increases by 3906 $\mu$s 256 times every second. Figure 4.4 shows round-trip delays computed during offset observations from ginger to larch. Each machine has a system clock with roughly 4 ms granularity, and the quantization of delay values is evident. The delay is zero if neither machine's clock advances during the measurement, or if they advance the same number of ticks. It is roughly 4 ms if the clock of the computer making the observation advances but the other computer's clock does not. Small variations from these values result from adjustments to the phase of the clock, as sometimes the effective value of the tick is slightly more or less than the nominal value.

Sometimes it appears that the system clock does not count ticks of the local oscillator as it should, but instead apparently misses some ticks. There are a number of possible causes. One is slow interrupt response — an operating system could fail to respond to a clock

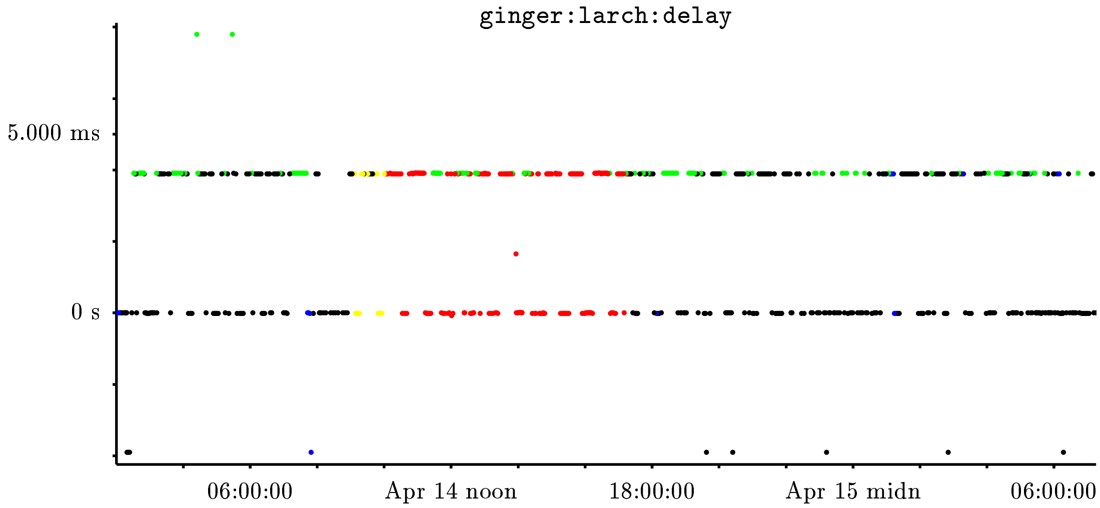



Figure 4.4: Delay to larch as observed by ginger.

interrupt for so long that a second would occur. Then, the interrupt handler would add the value of a tick to the system clock only once. Another cause is simply programming or design errors on the part of the operating system. A third, given that I am using the `rsadj` measurement technique, is that the additional programming required in the timekeeping daemon to keep track of corrections to the system clock might be erroneous, causing a misperception that the system clock did not keep time well, when instead there was a failure to keep track of some changes.

Figure 4.5 shows integrated predicted offsets of garlic from ginger. From examining other graphs, I know that, as usual, garlic was synchronized to far better than the 300 ms range of predicted offsets. Several features are evident in this graph. For most time periods, such as the one shown in detail, the predicted offsets show some noise, but appear to be locally reasonable. The most striking feature is that these periods of reasonable behavior are punctuated by rapid increases in the value of `poffset`. What appears to be happening is that the system clock on ginger loses an amount time not recorded for some reason by the `rsadj` instrumentation. When this happens, predicted offsets appear to increase in value, as the system clock has a smaller value relative to the remote clock after the loss of time than before.

## 4.4 Observation about remote clocks and networks

The basic `rsadj` technique completely dissociates the behavior of the synchronization algorithm and the behavior of the local oscillator, networks, and remote clocks. Earlier, I showed how to draw inferences about the behavior of the local oscillator by examining several remote clocks simultaneously. The question naturally arises as to how one might separate the effects due to networks and remote clocks. I consider in this section observations providing information about both of these effects, and later discuss how to separate them.

Figure 4.6 shows information about the delay distribution, delays, and piecewise predicted offsets to mizbeaver.udel.edu as observed by garlic. Note that while delays are gen-

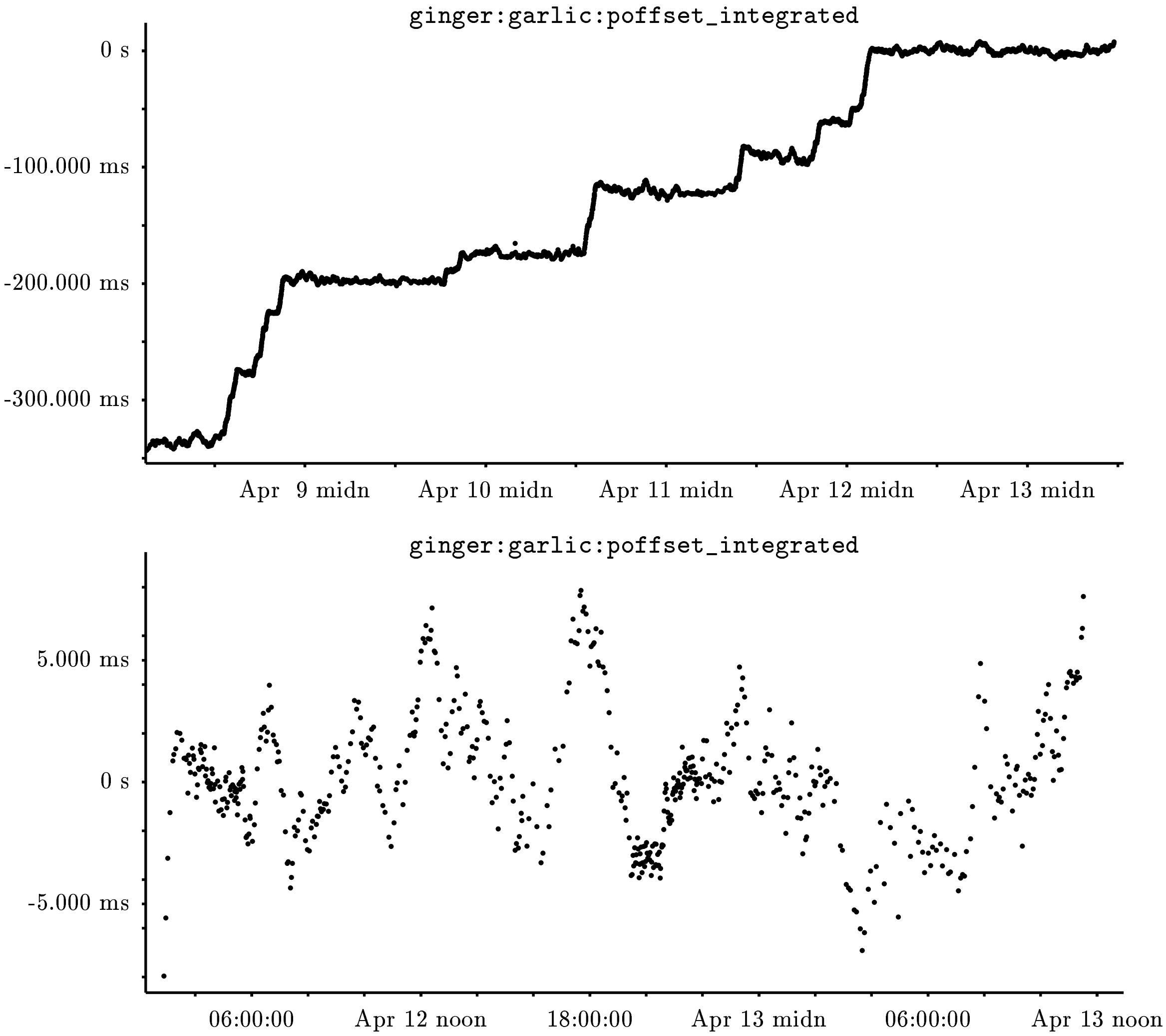



Figure 4.5: Integrated predicted offsets of garlic as observed by ginger. The bottom graph is a close-up view of approximately the last 36 hours covered by the upper graph.

erally low, occasionally very high-delay observations are recorded. These recorded observations really indicate groups of high-delay observations, since only the lowest-delay of the eight most recent observations is recorded. Examining the graphs together, one can see that many of the very high-delay observations correspond to large offsets.

Figure 4.7 shows a close-up view of the same data, and also shows the frequency estimates and intervals produced by the piecewise computation. As for pepper earlier, the frequency of the local oscillator is estimated to be different for different times. Unlike pepper however, the intervals cover periods of several days. Also, the difference in the frequency estimates is on the order of 0.03 ppm rather than 3 ppm.

From these graphs, I conclude that mizbeaver.udel.edu is a very well-behaved remote clock, and that it is being observed via a well-behaved network. The delay distribution has very few high-delay observations — even the 99th percentile is only 34 ms, less than

```
LOW DELAY  27.039 ms  HIGH DELAY 376.755 ms
1%      5%      10%     25%     50%     75%     85%     90%     95%     99%
27.94   28.30   28.52   28.84   29.31   30.03   30.50   30.90   31.62   34.29
p_base 28.305 p_add 29.312 p_mul 30.029 add 1.007 mul 0.060908
DELAY: ORIG 14591 OK 8246 INVALID 6345 (0.435 invalidated)
```

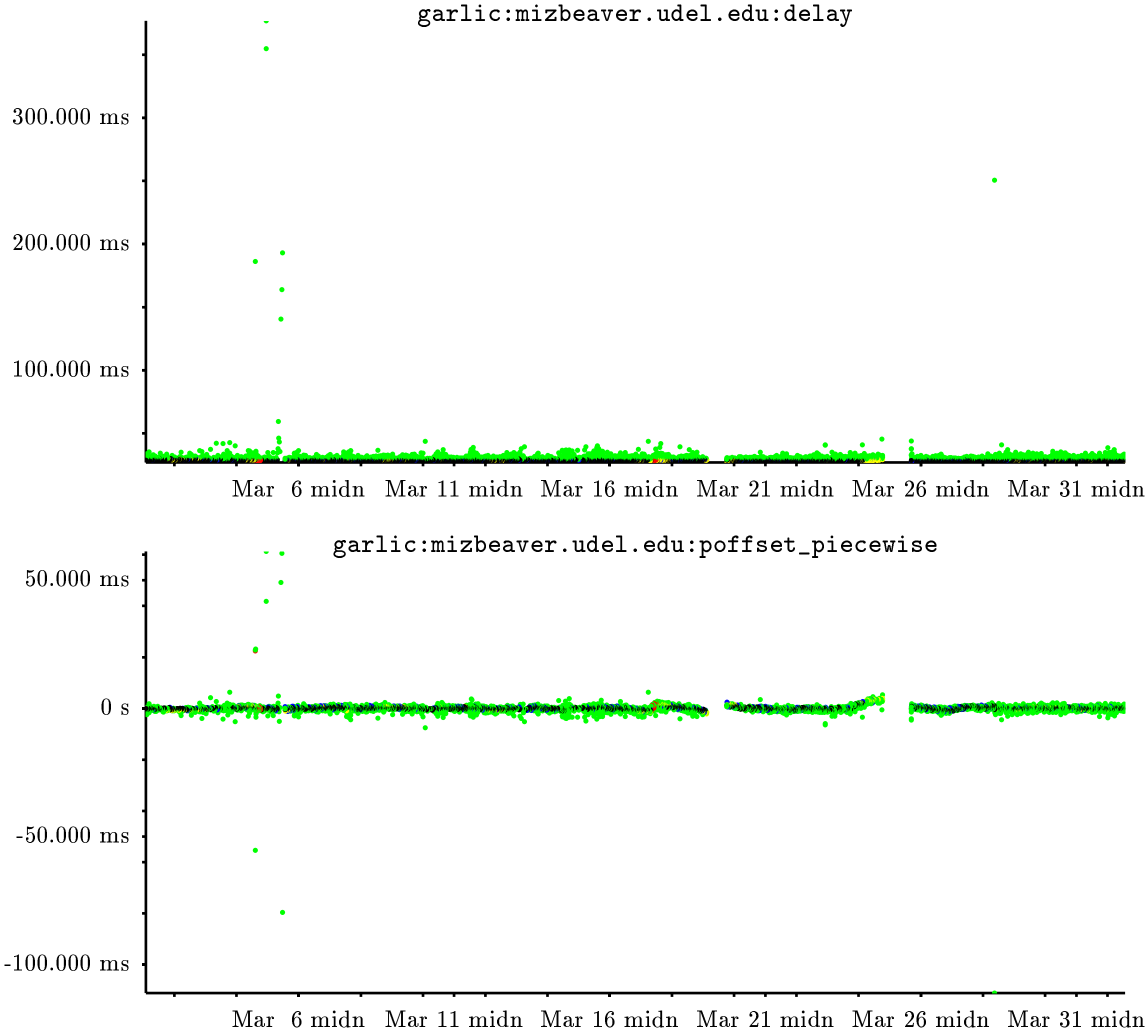



Figure 4.6: Information about the delay distribution, delay and piecewise predicted offsets of mizbeaver.udel.edu as observed by garlic.

10 ms greater than the lowest-delay observation. With the exception of some data near March 24, the values of `poffset` are quite low. However, examining delay data in that vicinity indicates that the average delay was higher, and thus more of the observations were marked invalid. Thus, the high values of `poffset` cannot be reliably attributed to errors in the remote clock — they could have been due to network delay variance or unmodeled frequency error in the local oscillator.

In fact, this type of artifact is fairly common with the `rsadj` technique — observations

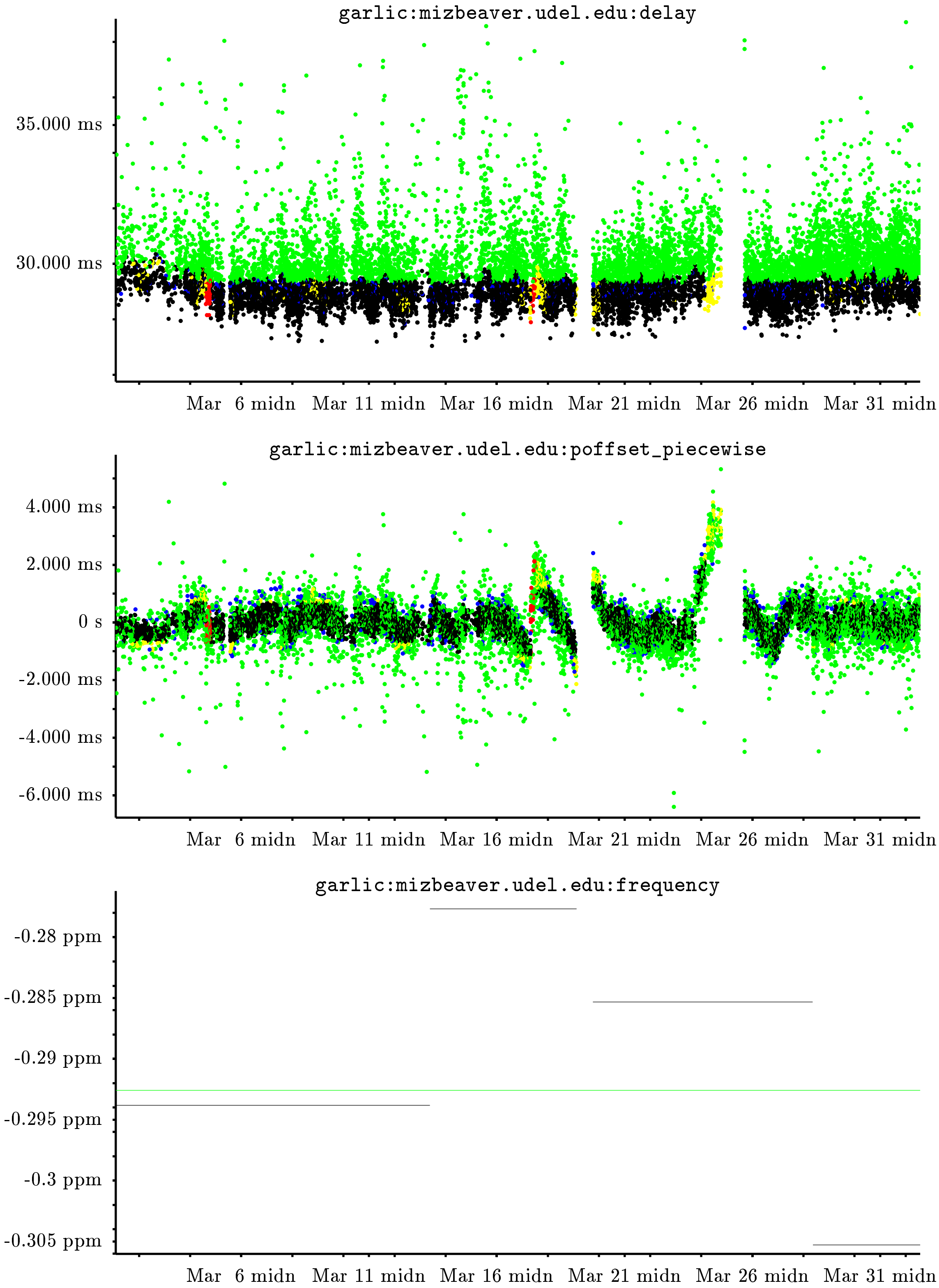



Figure 4.7: Delay, piecewise predicted offsets and frequency estimates of mizbeaver.udel.edu as observed by garlic. The green??? line shows the frequency estimate of the overall computation; the shorter lines are piecewise frequency estimates.

that have been marked invalid often appear to indicate errors that are in fact not fairly attributable to the source first indicated. Examining only valid observations avoids this problem, but also hides much useful information, such as the effects on `poffset` of high delays. Thus, when examining graphs, one should take into account all the data, but only draw conclusions about the local oscillator or the remote clock based on observations that have not been marked invalid.

Figure 4.8 shows overall predicted offsets, frequency estimates, and piecewise predicted offsets to psc.edu as observed by garlic. This remote clock, unlike mizbeaver.udel.edu and apple.com, does not have an attached reference clock, but instead synchronizes its clock by running NTP over the network to other remote clocks. Examining the graphs of overall predicted time and the frequency estimates from the piecewise computation, we can see that as in earlier graphs from pepper (Figures 4.1 and 4.2), the local oscillator appears to change frequency. However, I have already determined (Figure 4.3) that the local oscillator is more stable than the excursions observed in the offsets to psc.edu in Figure 4.8. Thus, I conclude that the variations in `poffset` in Figure 4.8 reflect the network and remote clock, rather than the local oscillator.

The delay distribution, shown in Figure 4.9, appears well-behaved, except for a small number of very low and high-delay observations. The 1st through 75th percentiles are tightly clustered, only 5 ms different, indicating that the vast majority of observations encountered roughly the same delay. Thus, the variations in predicted offsets to psc.edu are due to incorrect time on the remote clock, rather than network delay variance.

Figure 4.10 shows piecewise predicted offsets to pooch.osf.org as observed by garlic. Examination of the graph shows that the vast majority of predicted offsets are very close to zero. While a few of the predicted offsets that differ appreciably from zero — the ones near March 6 — do not appear to be part of any identifiable trend, a pattern of clustered negative-valued offset observations is evident at several times from March 7 to March 11. pooch.osf.org is usually synchronized to a WWV receiver, tracking HF timing signals from Colorado. I hypothesize that the negative-valued offset observations correspond to periods of time when pooch.osf.org was not receiving WWV signals properly.

One should check, however, to make sure that the negative-valued samples do not simply correspond to anomalies in the network. Examination of the graph of round-trip delays to pooch.osf.org from garlic covering the same period shows that the negative excursions do not in fact correspond to periods of high delay. Careful examination of Figure 4.10 reveals that many of the negative-valued observations are red, indicating that they were marked invalid because they are not self-consistent, rather than green, which they would have been had they been marked invalid due to having high delay.

During the time that data were being gathered for this research, the maintainer of umd1.umd.edu, a computer providing NTP time service via an attached WWVB receiver, announced that on Sunday, March 27, 1994, the LSI 11/73 would be replaced by a Sparc 10 and that the address would change. Observations of umd1.umd.edu and of the new computer at the new address were made by garlic. Figure 4.11 shows the round-trip delay observed to the umd.edu timeserver, and also shows integrated predicted offsets of the two machines from garlic.

The left vertical line marks the time at which the delay to umd1.umd.edu appears to have changed; I added it to the graph based on the observed values of delay. The right

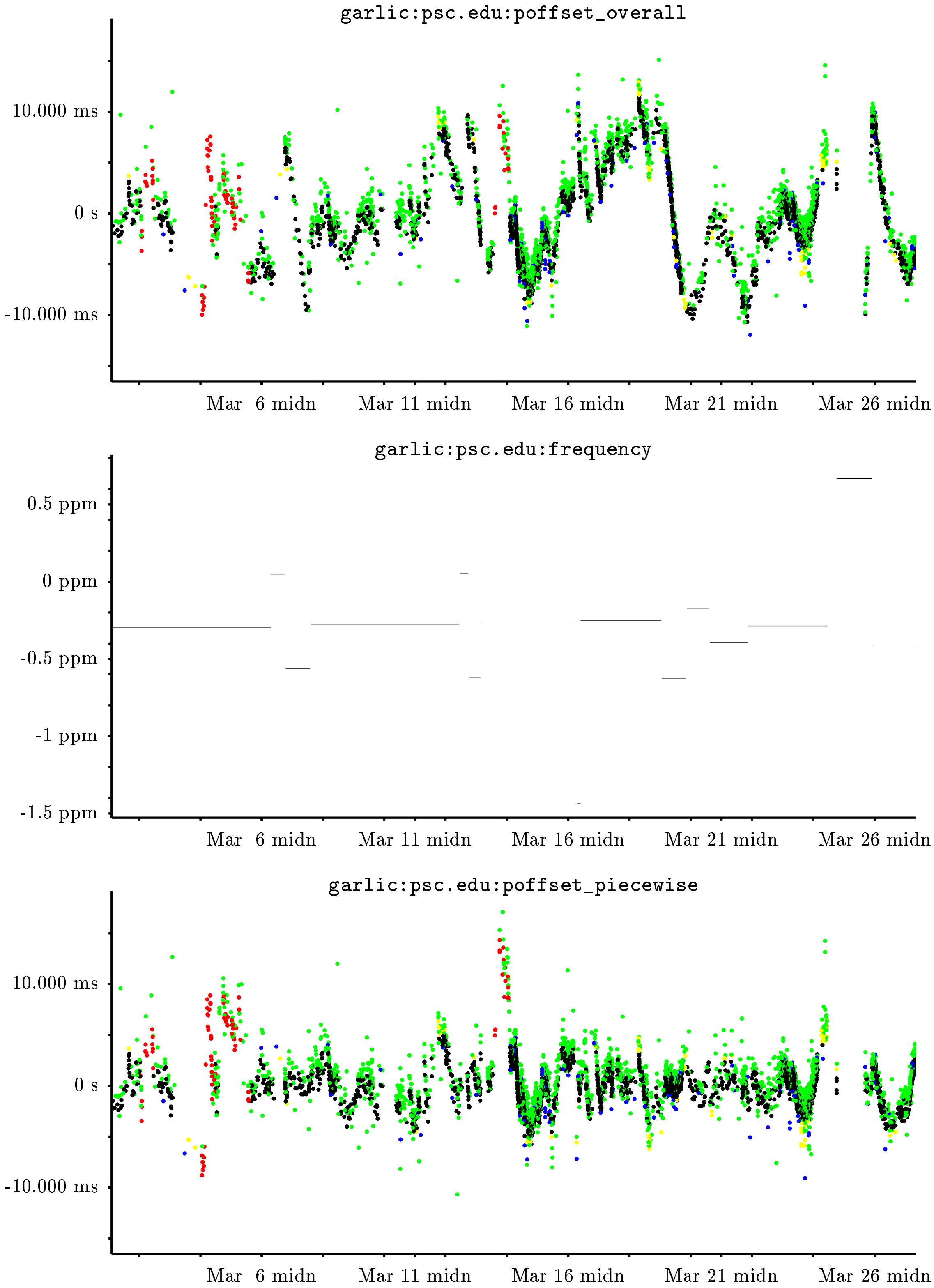



Figure 4.8: Overall predicted offset, frequency estimates and piecewise predicted offset to psc.edu from garlic.

```
LOW DELAY  18.219 ms  HIGH DELAY 201.645 ms
1%      5%      10%     25%     50%     75%     85%     90%     95%     99%
26.21   27.27   27.74   28.61   29.77   31.07   31.84   32.47   33.88   39.60
p_base 27.267 p_add 29.770 p_mul 31.067 add 2.503 mul 0.139363
DELAY: ORIG 4211 OK 2407 INVALID 1804 (0.428 invalidated)
```

Figure 4.9: Statistical characterization of round-trip delays from garlic to psc.edu.

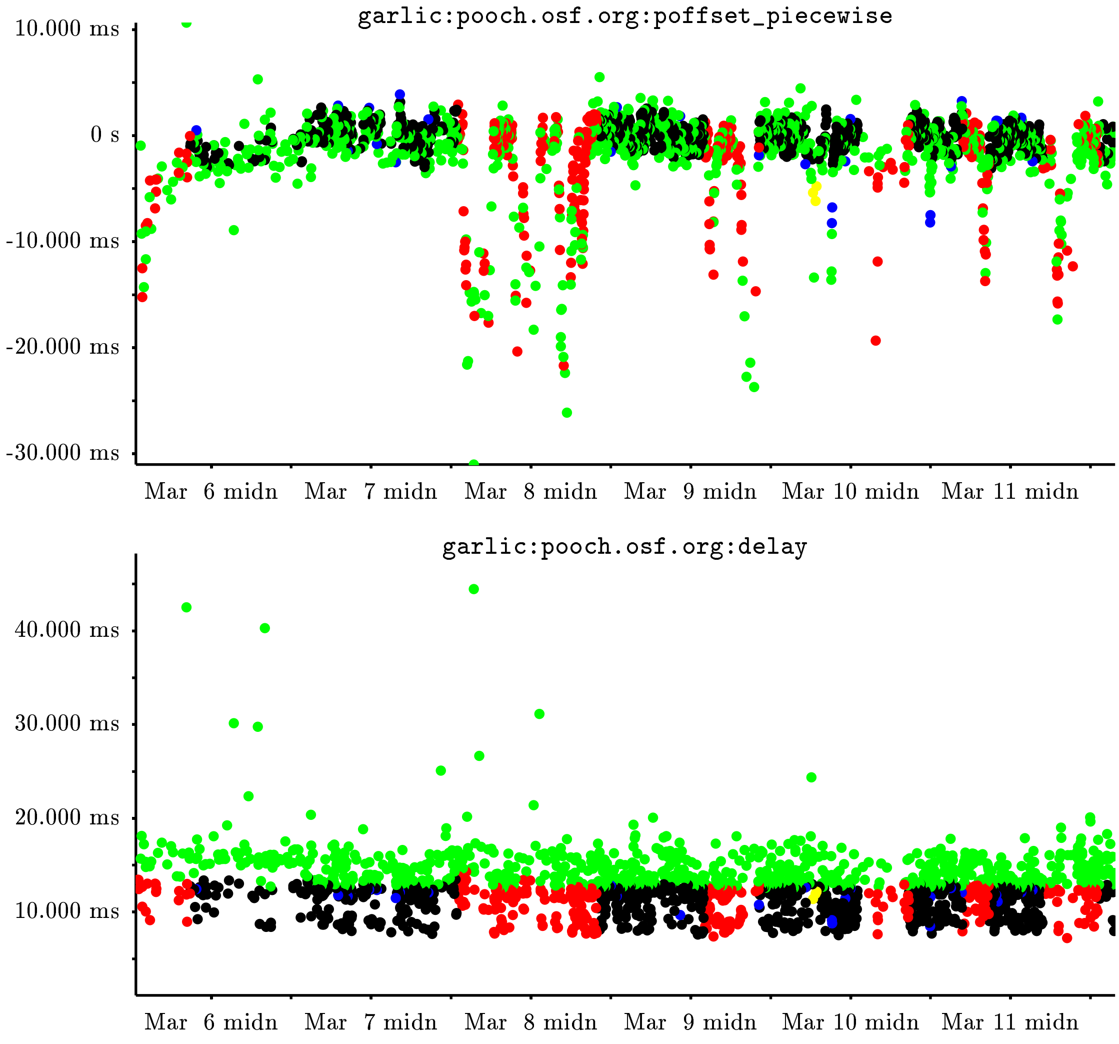



Figure 4.10: Piecewise predicted offsets of pooch.osf.org, top, and delay, bottom, as observed by garlic.

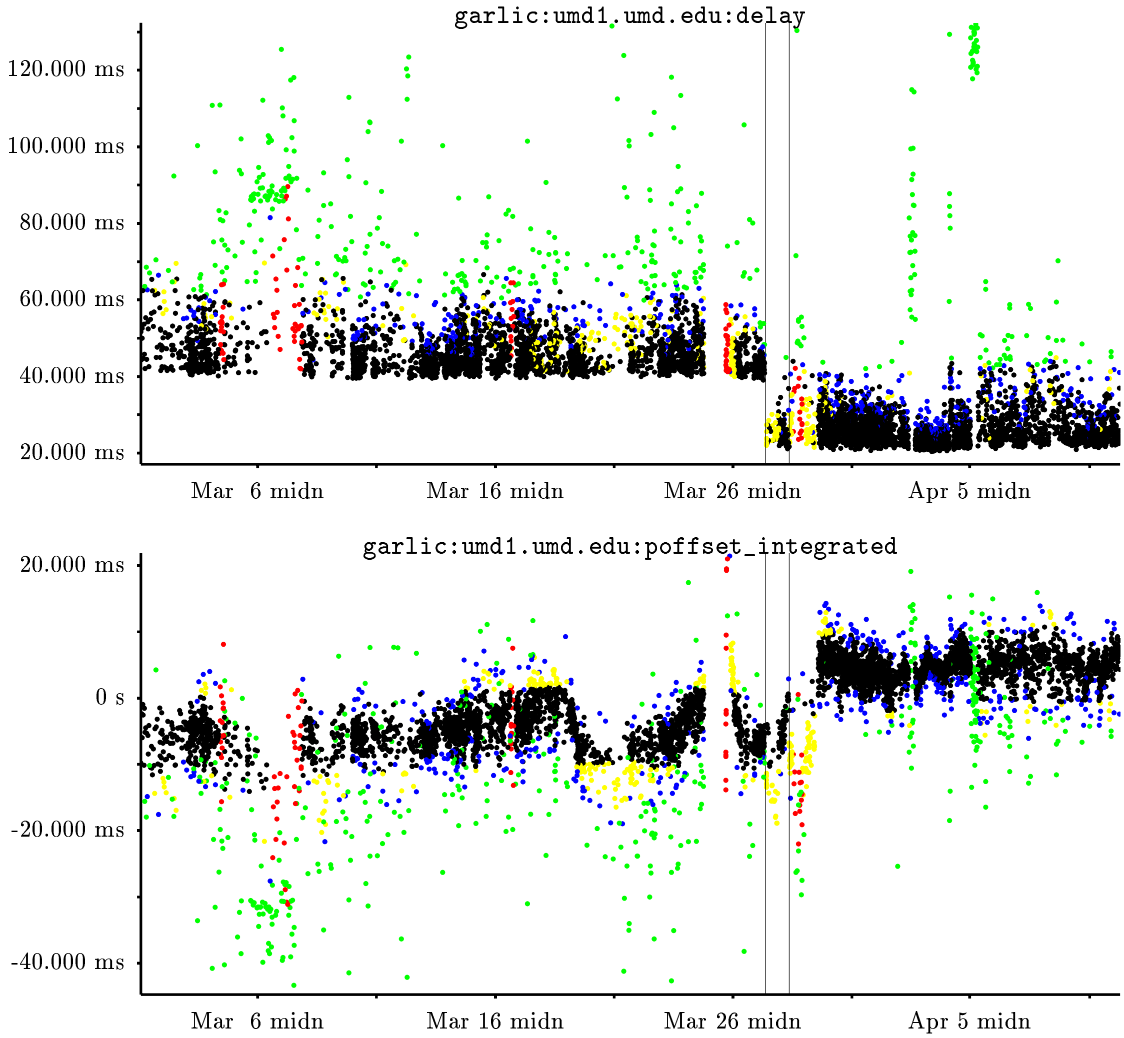



Figure 4.11: Delay and integrated predicted offset of umd1.umd.edu/haven.umd.edu as observed by garlic. The left line corresponds to the observed change in delay, and the right line to the time after which observations are from haven.umd.edu.

vertical line marks the boundary between recorded offsets to 128.8.10.1, the address of the old computer, and recorded offsets to 128.8.10.6, the address of the new computer, known as haven.umd.edu. To produce these plots, I modified the log messages to make these two computers appear as one. That is, before running the analysis program, I temporarily changed all values of 128.8.10.6 to 128.8.10.1. I did this in order to undo the change of address so that data from the same logical time server could be examined before and after a change of hardware and IP address.

Several features of these graphs are striking. Perhaps the most notable is the sudden drop in round-trip delay, which I have marked by the left vertical line. Another is that there appears to have been a change in systematic offset roughly when garlic began recording observations from the new IP address. The use of integrated predicted time allows us to see this effect relative to the other remote clocks, since phases of each run are estimated

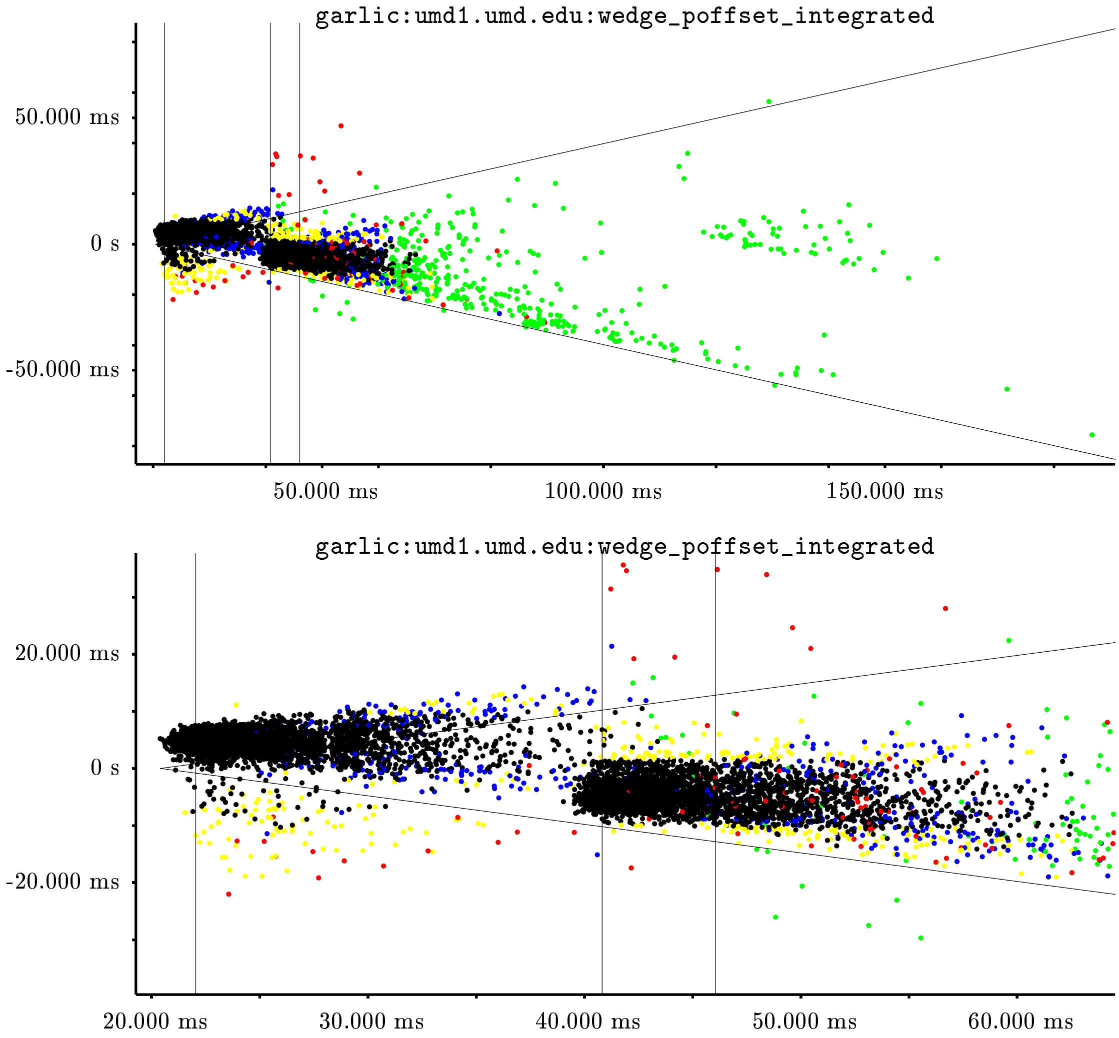



Figure 4.12: Scatter plots of integrated predicted offset vs. delay for umd1.umd.edu/haven.umd.edu as observed by garlic. The vertical lines mark the 5th, 50th and 75th percentiles of delay for the underlying data set. The wedge has its vertex at zero offset and the minimum delay, and has slope 1/2, corresponding the offset errors expected from maximally asymmetric additional delay beyond the minimum.

based on data from multiple remote clocks. Thus, a change in systematic offset is apparent. If overall predicted time for this remote clock were used, the change in systematic offset would be reflected in differing estimates of the phases for runs taking place before and after the change, rather than being apparent in the predicted offsets.

Another view of the data is helpful in understanding the changes that occurred. Figure 4.12 shows a scatter plot of integrated predicted offset vs. delay, and also a closer view of the same data. In the upper plot, three wedge-shaped groups of observations are apparent. The leftmost one corresponds to the rightmost portion of the previous figure. The middle wedge corresponds to the leftmost portion. The wedge at the right, has far fewer observations, and is harder to place. Upon close inspection of the graph of delay vs. time in

Figure 4.11, it appears that it corresponds to a period of temporarily elevated delay near April 5.

The lower graph shows more detail of the left and center wedges. Here the change in systematic offset can be clearly seen. It is interesting to speculate on the reasons for these changes. Perhaps the PDP 11/73 was suffering from queuing delays in processing packets. If the packets were queued before the incoming timestamp were taken, the packets would suffer apparently excessive delay on the outbound trip, which would appear as a incorrect positive offset. However, if one assumes that the offsets corresponding to the faster processor and lower delay are correct, the systematic offset of the group of high-delay samples appears negative. Perhaps, then, the packets are timestamped upon receive, timestamped for transmitting, and encounter delay in an output buffer, thus causing extra delay on the return trip.

The upper graph in Figure 4.12 has many observations along the lower line of the wedge superimposed on the data, corresponding to observations encountering additional delay mainly on the return trip. Note that the "return trip" consists of all actions after the second timestamp is taken at the remote computer and before a timestamp is taken when the packet is received back at the computer making the offset measurement. The graph of delay in Figure 4.11 shows that in addition to the delay values being generally greater on the left side of the graph, the variation in delay is also greater. These observations support the hypothesis that umd1.umd.edu appears to have had a negative systematic offset due to output queuing delays, as one would expect output queuing delays to cause greater variance in return-trip delays.

Figure 4.13 shows scatter plots of offset vs. delay for several remote clocks. The particular type of offset shown is $\mathtt{poffset}_{\mathtt{integrated}}$. Thus, the graph indicates not the behavior of the remote clock as observed by the NTP algorithms, but the behavior of the remote clock according to the local oscillator. Such graphs are useful to examine, in that they give information about the stability of the local oscillator and remote clock, and also about the characteristics of the network delay behavior. On each graph, vertical lines mark the 5th, 50th and 75th percentile of the delay distribution. The lines forming a wedge begin at zero offset and the smallest observed delay, and have slope 1/2, corresponding to the maximum error that can be induced by a given amount of extra delay.

The top graph is a close-up view of observations made to mizbeaver.udel.edu. While there are several milliseconds of variation in delay, note that the variation in offset for samples near the lowest delay is much smaller than 1 ms. Very few points fall outside of the wedge, and the observations mostly fill the interior space, indicating that the local oscillator was modeled well and that the value of time on mizbeaver.udel.edu was correct through the period. The second graph shows offsets to apple.com, but with a view also showing the high-delay observations. Note that they are near the upper wedge line, indicating that the packets encountered extra delay on the outbound trip. The bottom graph show offsets made to err-gw.ethz.ch, located in Switzerland. A few observations lie well outside the wedge, and careful inspection reveals that the two low-delay observations with values near 20 ms have been marked invalid by the computation of predicted time. While other observations also have high offsets, they have high delays, and thus do not indicate that the remote clock had the wrong time. The two high-offset low-delay observations, however, indicate some kind of problem.

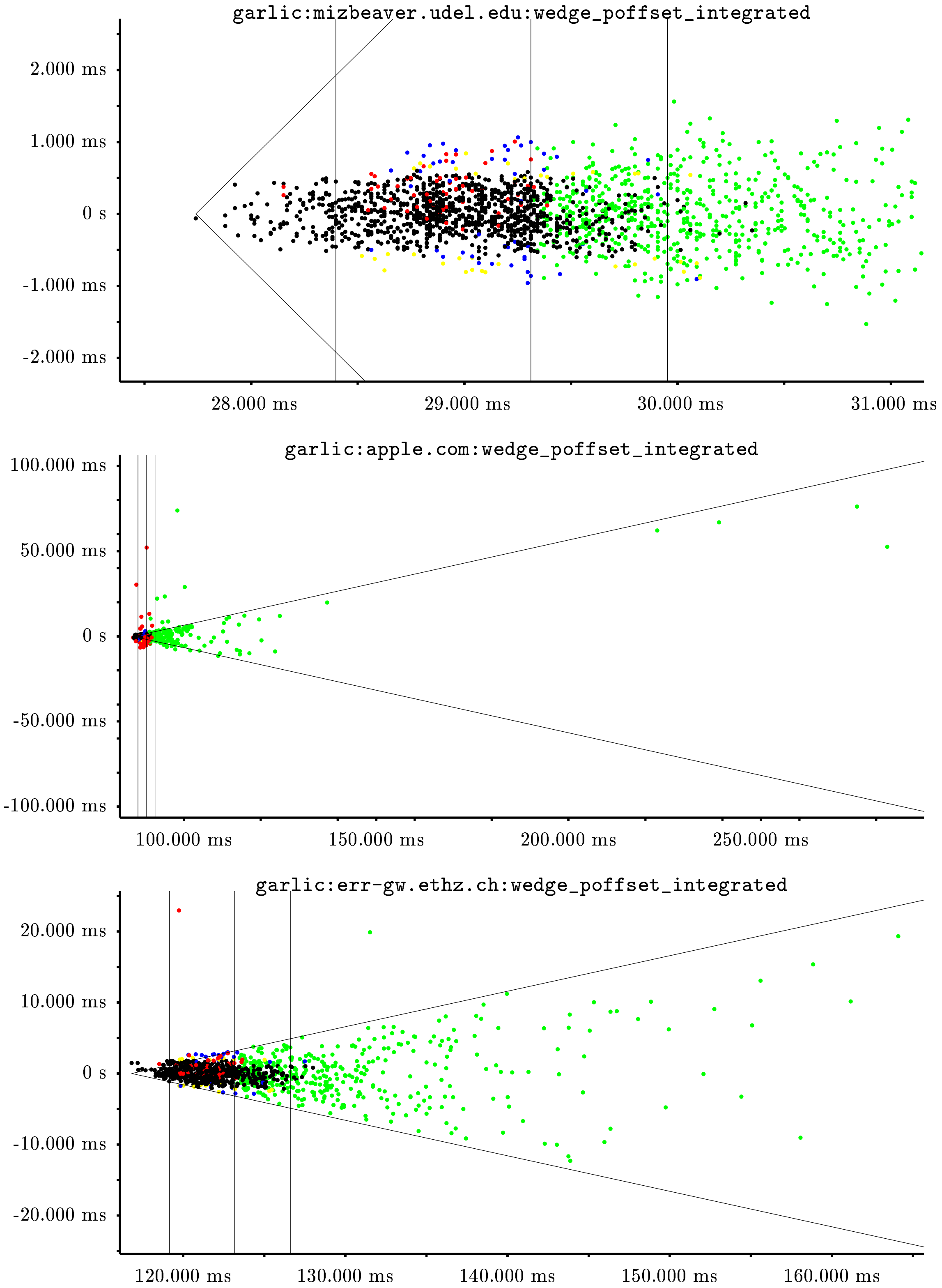
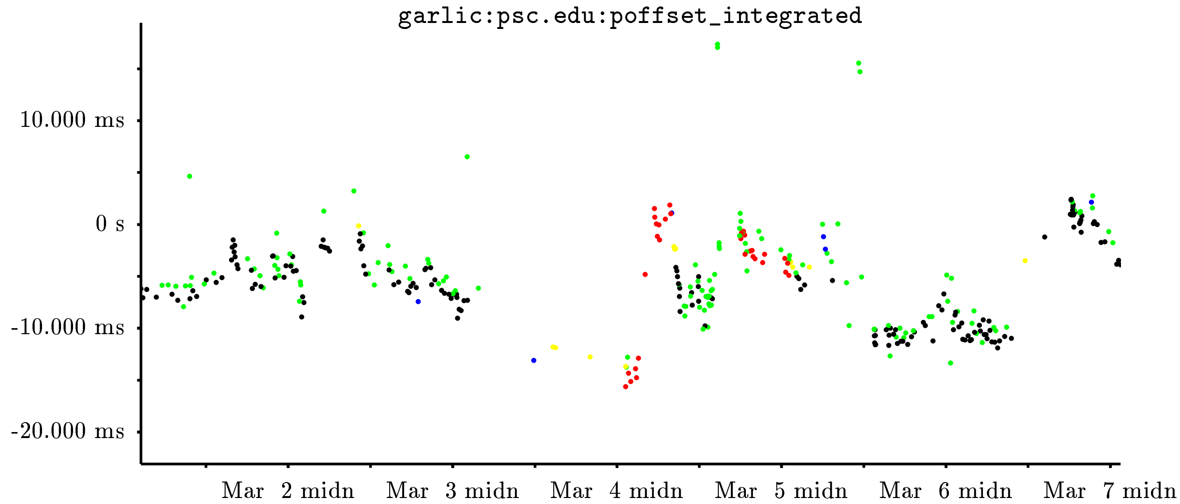



Figure 4.13: Scatter plots of integrated predicted offsets to three remote clocks vs. delay as observed by garlic over 6 days starting March 1, 1994.



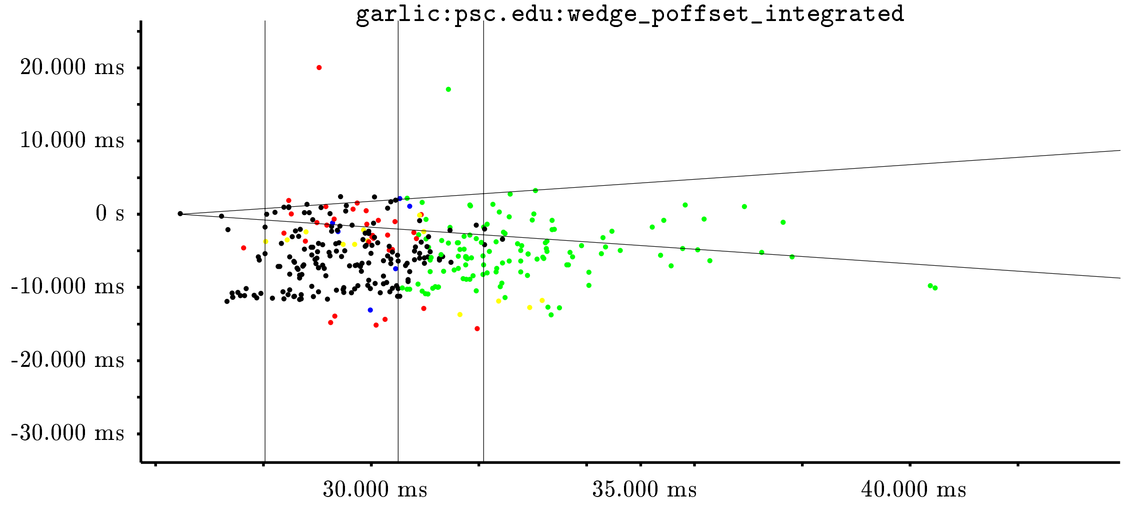




Figure 4.14: Integrated predicted offsets vs. time to psc.edu from garlic, and a scatter plot of the same offsets vs. delay.

Figure 4.14 shows two views of integrated predicted offsets to psc.edu. The top graph shows them plotted as a function of time. While there might appear to be some waviness to the data, indicating that either the local oscillator or the remote clock is not stable, it is not strikingly clear, as some earlier graphs were. However, the scatter plot of offset vs. delay makes the lack of stability quite clear. In contrast to the previous graphs, low-delay observations exhibit great variation in offset. Because the local oscillator was in fact stable, this variation can be attributed to wander in the time of the remote clock. Earlier, Figure 4.8 showed this lack of stability at psc.edu, but the scatter plot allows it to be seen more clearly.

### 4.4.1 Systematic offsets

The `rsadj` technique can be used to observe the relative systematic offsets of the various

```
aging 0.000000 ppm/day freq -0.291707 -0.291707 ppm
Run 0   -0.087470 s
Run 1    0.114431 s
[      128.4.1.2] # 0 offset   3.601 ms rms error 0.318 ms weight 9.878e-01
[     128.8.10.1] # 1 offset -13.367 ms rms error 1.984 ms weight 9.79e-01
[     130.43.2.2] # 2 offset -13.257 ms rms error 0.874 ms weight 6.088e-01
[  130.105.1.156] # 3 offset  -0.546 ms rms error 1.078 ms weight 2.677e-01
[    129.116.3.5] # 4 offset   4.520 ms rms error 1.276 ms weight 9.851e-01
[    127.127.4.1] # 5 offset -14.130 ms rms error 0.642 ms weight 9.726e-01
```

Figure 4.15: Sample output from the program implementing the computation of integrated predicted time. Data are offsets observed by garlic for 72 hours beginning March 7, 1994, 0700 UTC.

remote clocks being observed. Recall that when computing integrated predicted time, a frequency, possibly an aging rate, one phase per run, and one systematic offset per remote clock is estimated. While it cannot be determined which of the remote clocks is correct, the indication of relative offset is still useful.

Figure 4.15 shows the estimated parameters for offsets observed by garlic; aging was not estimated, and the time period under study contains two runs, numbered 0 and 1. Each remote clock is listed, along with the estimated systematic offset, rms error of the clock's offsets with respect to the estimation over all remote clocks, and that clock's weight in the integrated estimation. Recall that integrated predicted time is based on offset observations for each remote clock, and uses those observations contained in the last interval generated by the piecewise computation. I examined further output, not shown, and found that for each remote clock except 127.127.4.1, an attached WWVB receiver, data were considered over the entire 3-day period. For 127.127.4.1, only 6 hours of data were considered.

Examination of the estimated offsets and rms errors shows that variation in offsets is very large compared to the values of rms error. If the variation in offsets were small compared to the values of rms error, one might think that the values of systematic offset did not reflect actual systematic offsets, but were the results of offset measurement noise. Given this data, however, I must conclude that there are significant *perceived* systematic offset errors among the various remote clocks. The word perceived is important; I cannot attribute these errors specifically to asymmetric network delays, or to systematic biases in the actual time on the remote clocks, as they could be due to either cause.

## 4.5 Observations of reference clocks

I can use the `rsadj` technique to observe locally-attached reference clocks in addition to remote clocks accessed via a network. The only significant difference is that rather than using `delay` as an *a priori* indicator of erroneous offset measurements, I use the error indication computed by the NTP daemon, which is intended to indicate the reliability of the offset measurements from the reference clock. As explained previously in Section 3.4.2, the computation of piecewise predicted time processes the error indications as if they were delay values.

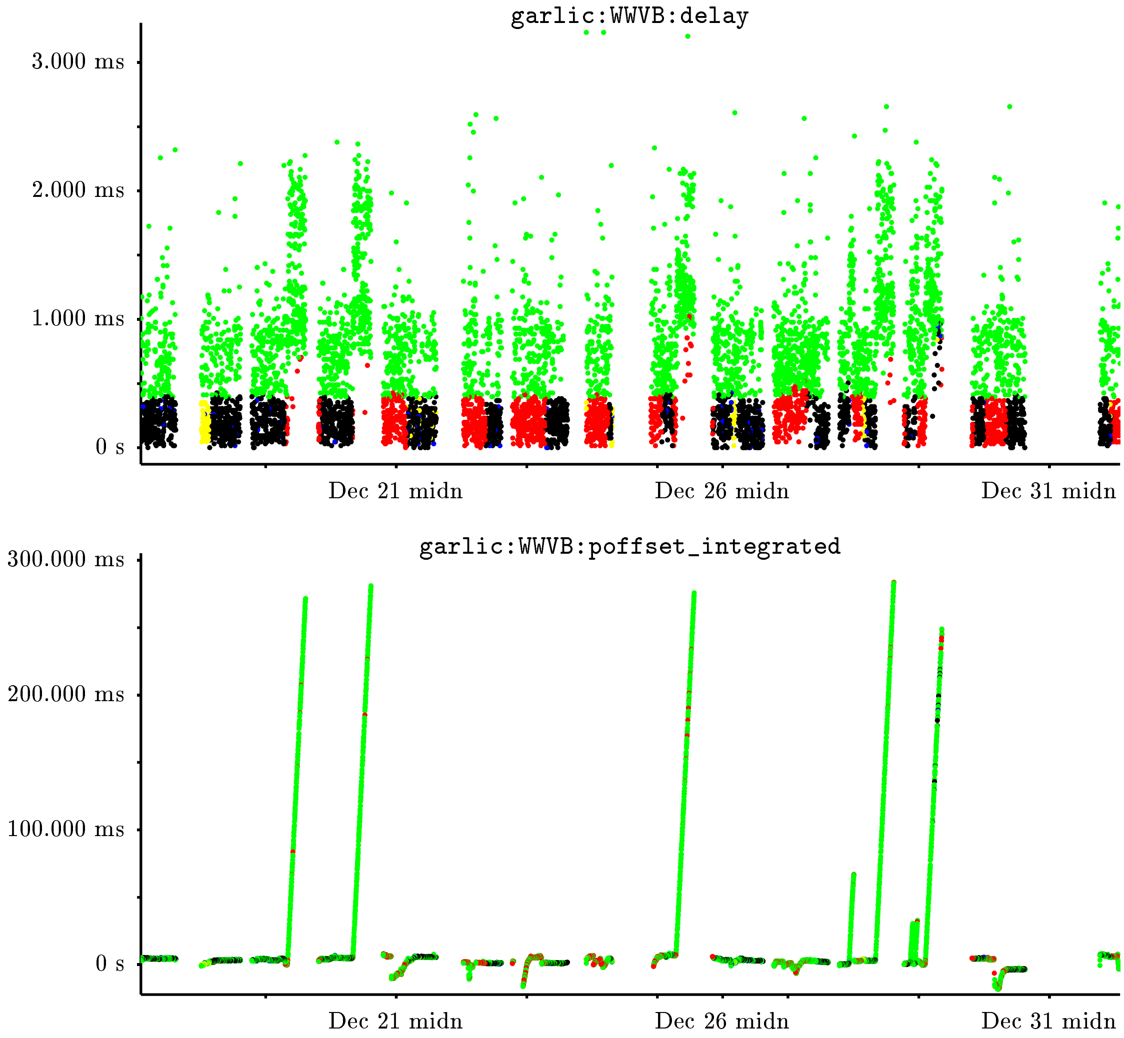



Figure 4.16: Error indications and integrated predicted offsets of the WWVB receiver attached to garlic during December 1992.

### 4.5.1 Spectracom WWVB receiver

A Spectracom model 8170 WWVB receiver was attached to garlic. Using the `rsadj` technique, I have examined the behavior of this reference clock. I have found that at times the clock does in fact appear to be phase-locked to the WWVB signal, but that at other times it appears to lose lock, i.e., fail to accurately track the phase of the received signal, and return increasingly erroneous time. Also, the receiver appears to lock to WWVB with various phase errors.

Figure 4.16 shows error indications, which the `rsadj` technique interprets as delay, and integrated predicted offsets for the WWVB receiver. Integrated predicted offsets, rather than piecewise or overall, were chosen in order to give a view of the WWVB receiver's behavior relative to the estimated behavior of the local oscillator. Had overall or piecewise predicted offsets been used, the estimates of the local oscillator's behavior would have been

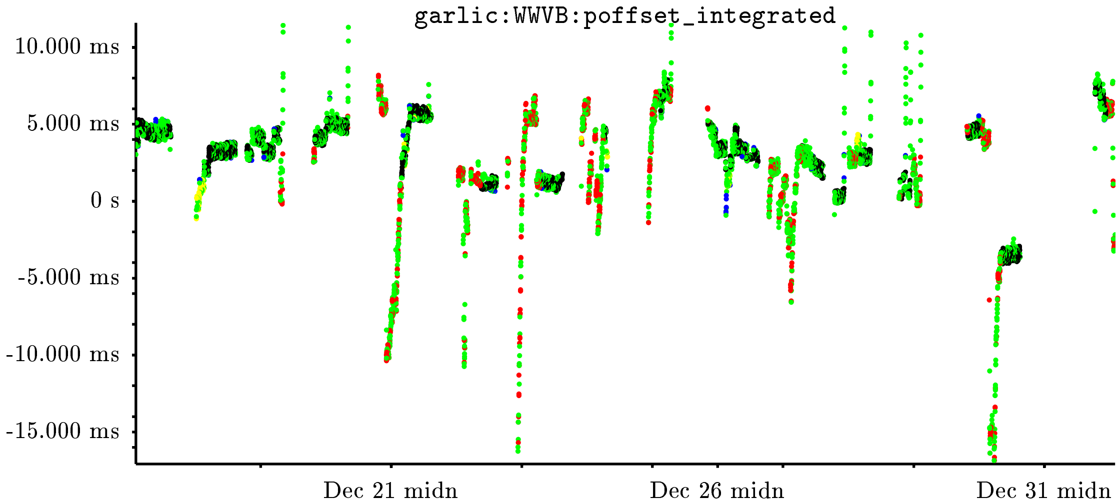



Figure 4.17: Closeup view of integrated predicted offsets of the WWVB receiver attached to garlic.

derived from the observed offsets of the WWVB receiver. Since the offsets observed from the WWVB receiver in fact contain many outright incorrect values, my assumption that the remote clock has the correct time is incorrect and thus the estimates would likely be (and in fact were) erroneous.

Figure 4.16 shows that on five occasions the observed offsets to the WWVB receiver climbed steadily and became very large, over 200 ms. Note, however, that on each occasion that rate of change of the predicted offset values appears consistent. I hypothesize that this reflects the natural rate of the oscillator in the receiver which is normally phase-locked to the received signal. On several other occasions, observed offsets climbed to approximately 50 ms, but then became closer to 0 s. The graph of delay indicates that these excursions correspond to times when the error indication from the receiver was elevated.

Figure 4.17 shows a close-up view of integrated predicted offsets to the WWVB receiver for the same time period as the previous figure. During some time periods the receiver appeared to return stable time indications, but it appeared to have different systematic offsets relative to the local oscillator during these different time periods.

Figure 4.18 shows error indications and integrated predicted offsets for the same receiver for a different time period. Significant portions of this data appear well-behaved; on February 21 and during the first part of February 24, the predicted offsets appear to have only small amounts of noise, and do not exhibit a constant change over time. Assuming that these periods are "good" receiver behavior, this gives us confidence that the estimates of local oscillator behavior used to compute these values of `poffset` are correct. However, during other time periods the receiver appears to wander. Even the two "good" periods identified earlier have different phases.

Figure 4.19 shows a close-up view of the integrated predicted offsets to the WWVB receiver. My hypothesis explaining this behavior is that the receiver undergoes cycle slips where phase-lock is lost briefly. In the cases on the left side of the graph, it appears that one or two milliseconds is lost each time. On the right, it appears that the internal receiver oscillator may be operating unlocked. The rate of increase of `poffset` is 10 ms in 4 hours,

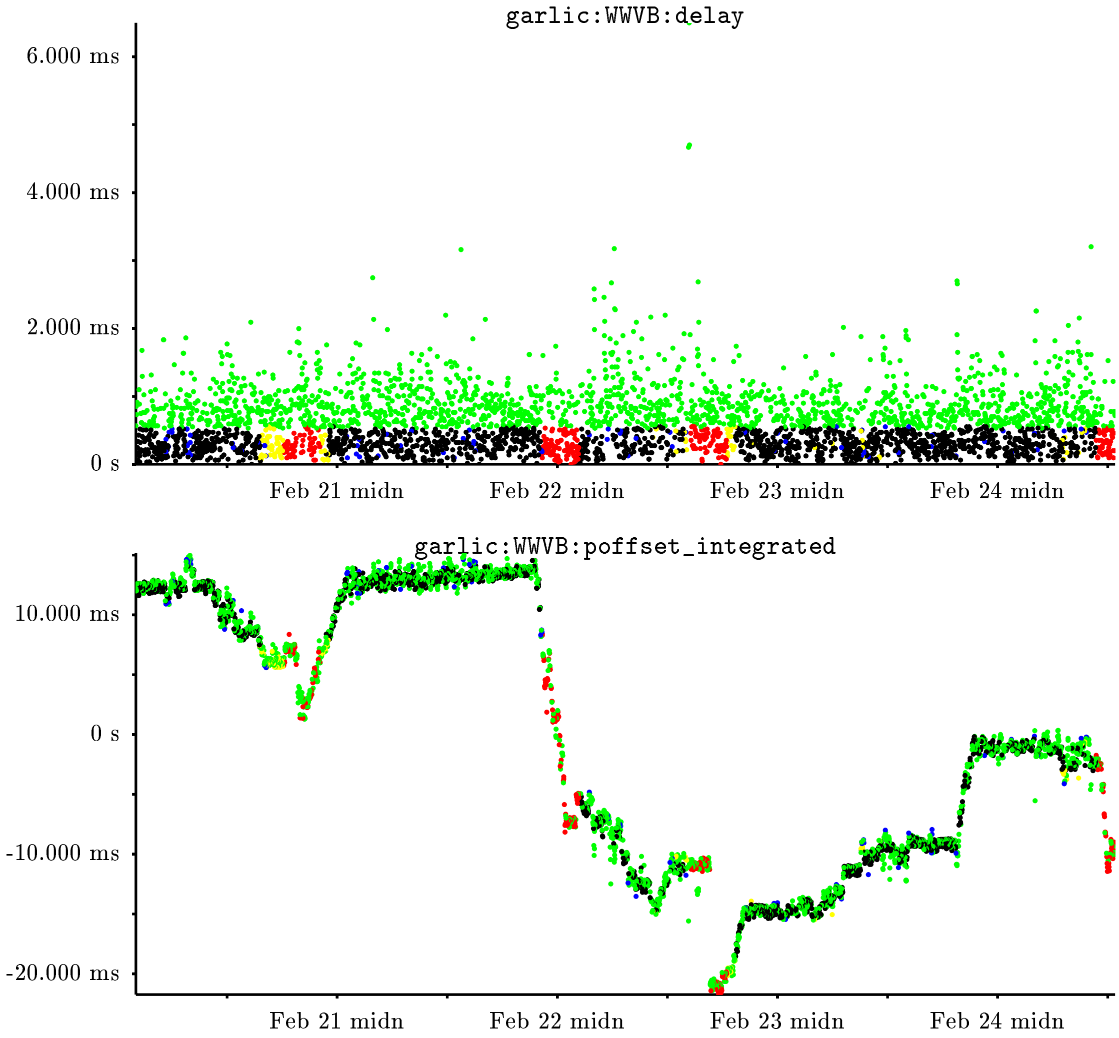



Figure 4.18: Error indications and integrated predicted offsets of the WWVB receiver attached to garlic during February 1994.

or roughly 0.7 ppm.

From examining these graphs, and many others like them, I conclude that this particular receiver is not useful for precision timekeeping given the low signal levels and high noise levels presented to it. While the receiver might well exhibit improved behavior with a greater signal level, it exhibits a great deal of wander while returning small error indications.

### 4.5.2 Magellan GPS Receiver

The `rsadj` technique can be used to examine the results of observations of the GPS receiver attached to ipa. While I later use GPS as an external trusted reference, it is also interesting to observe the behavior of the GPS receiver. Such observations increase confidence in the overall accuracy of the indicated times from the GPS receiver, while pointing out that a few samples are likely grossly incorrect.

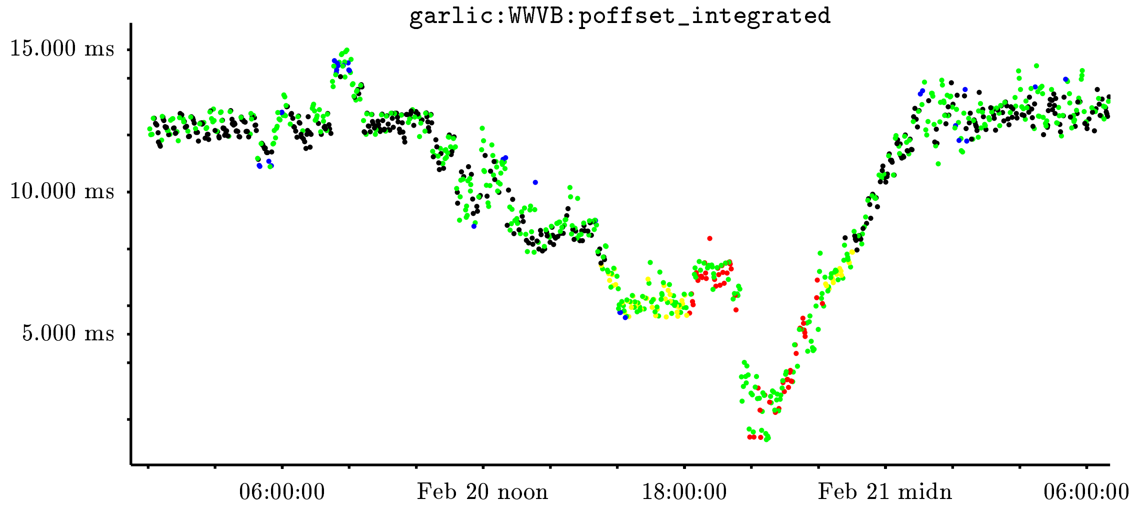



Figure 4.19: Closeup view of integrated predicted offsets of the WWVB receiver attached to garlic.

Later, when I discuss the use of GPS-derived time to verify the validity of the `rsadj` technique, I will explain how values are obtained which are in effect offset observations of the GPS receiver. For now, assume that offset observations have been made as if it were a reference clock. I will call those observations $\texttt{aoffset}_{\text{GPS}}$, or simply `aoffset` in this section.

Figure 4.20 shows recorded actual offsets of the GPS receiver as observed by ipa, and integrated predicted offsets. While some values of `aoffset` are large, on the order of 50 ms, the corresponding values in the graph of `poffset` are much smaller, no greater than a few milliseconds. While the graph of `poffset` shows a wavy trend line, the values of `poffset` never differ greatly from the short-term trend. If there were reason to believe that the local oscillator of ipa was more stable than the variations indicated by this data, there would be cause for concern about the accuracy of the GPS receiver. However, this data appears fairly typical of the local oscillator in ipa. The predicted offsets to garlic closely match those to the GPS receiver; if the local oscillator were accurate and the GPS receiver wandering, this would not be the case.

Figure 4.21 shows frequency estimates of the local oscillator of ipa and piecewise predicted offsets to the GPS receiver based on those estimates. If the GPS receiver is assumed correct, the frequency of the local oscillator of ipa varies somewhat over periods of several hours to a day or so. The piecewise predicted offsets are generally quite small, and have low rms error. At times they appear to begin departing from zero in a smooth manner, but closer inspection reveals these observations to have been marked invalid for one reason or another; thus the estimation process does not consider them.

The short-term stability of the GPS receiver appears excellent, and given the known lack of long-term stability of ipa's local oscillator, together with the correlation of GPS time and observations of garlic, this data cannot be used to argue that the GPS receiver lacks long-term stability.

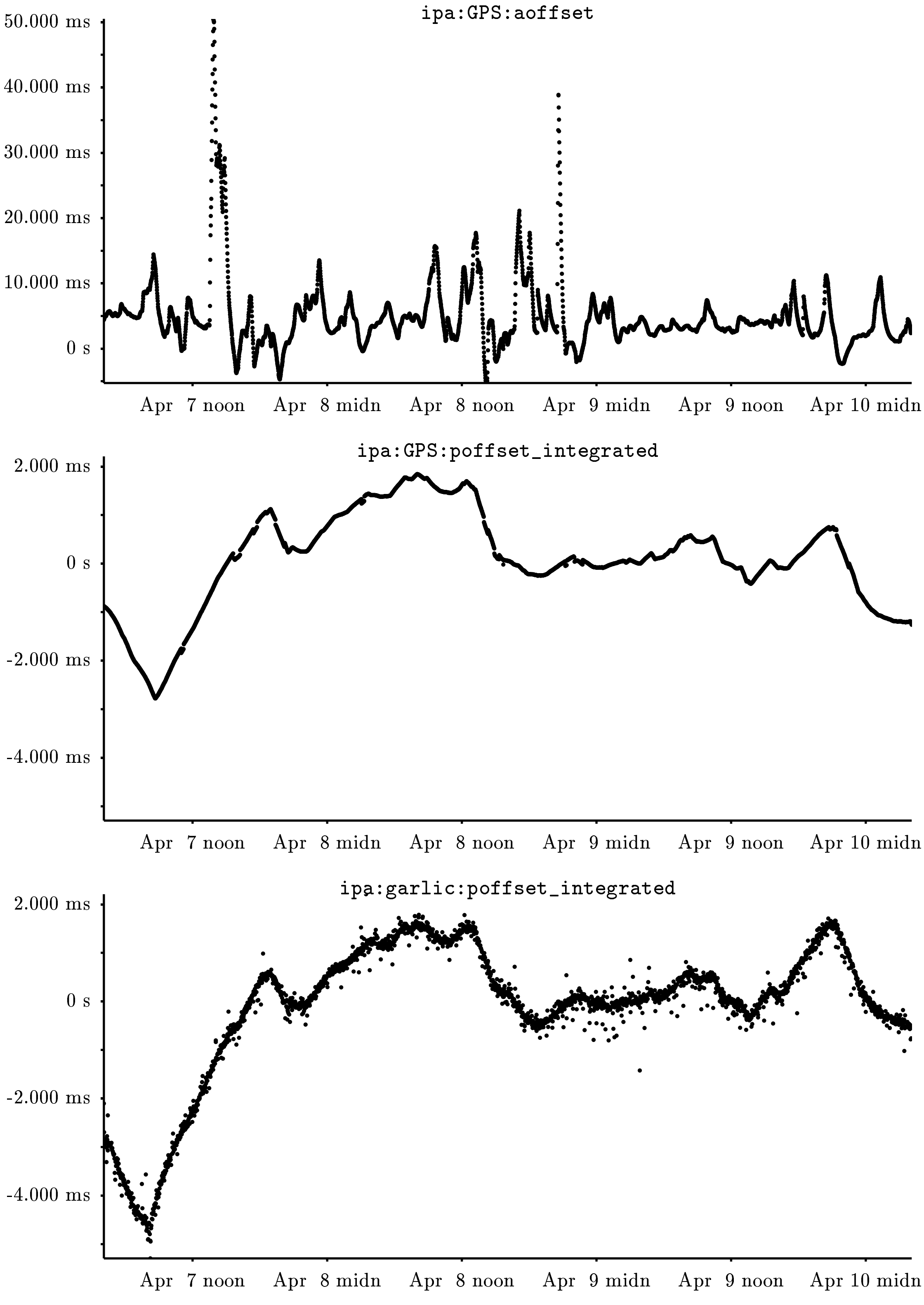



Figure 4.20: Actual offsets and integrated predicted offsets, top and middle of the GPS receiver as observed by ipa during April, 1994. Integrated predicted offsets to garlic, bottom.

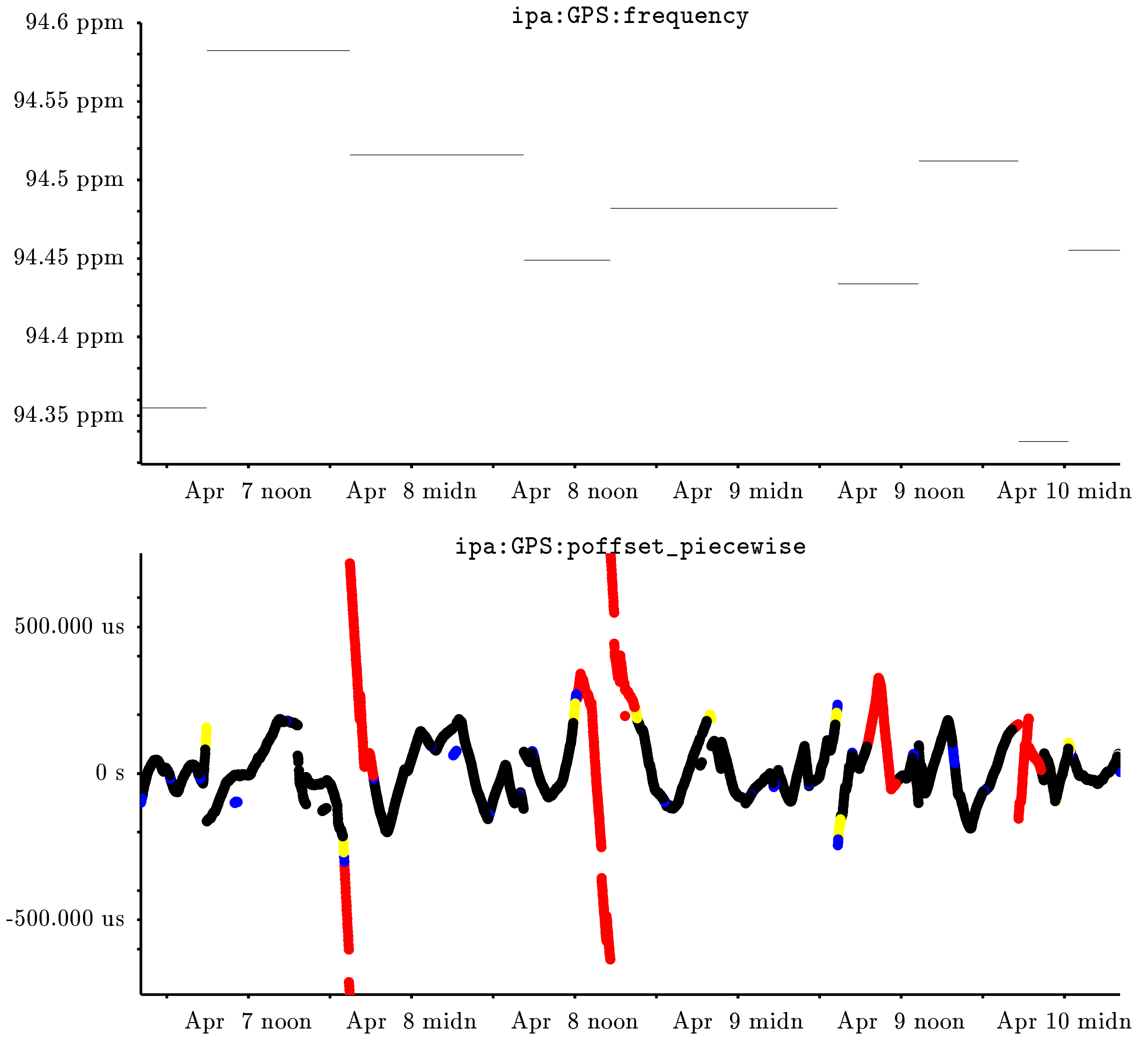



Figure 4.21: Frequency estimates and piecewise predicted offsets of the GPS receiver as observed by ipa.

## 4.6 Observation of NTP's behavior

Earlier in this chapter, I examined the behavior of the local oscillator, networks, remote clocks, and attached reference clocks using the `rsadj` synchronization measurement technique. I examined graphs of quantities derived solely from values `uoffset` — various forms of `poffset` and frequency estimates based on uncorrected offsets. Now, I will also examine quantities derived from `rsadj`, specifically various forms of `rsadjresid`. Rather than conveying information about remote clocks and networks, these quantities reveal how the existing synchronization algorithm — generally the Network Time Protocol — actually did control the behavior of the system clock.

In general, NTP works by adjusting the system clock in such a way as to keep the offsets it observes, labeled `aoffset` in my scheme, small. In addition to adjusting the phase of the system clock, NTP maintains an estimate of the frequency error of what I have termed the local oscillator. This estimate is also adjusted to keep values of aoffset small; if aoffset is consistently positive, the frequency error estimate will be increased, causing a higher effective rate of the system clock, causing values of aoffset to become more negative.

Figure 4.22 shows three graphs that tell us much about the behavior of the NTP algorithms[1] running on garlic during the indicated time period. This time period was chosen because it contained various features that show various typical behaviors. The top graph shows the values of NTP's frequency estimate; the thin line shows the frequency estimated by the rsadj technique. The middle graph shows actual values of NTP's system offset, the values resulting from NTP's offset measurement operation.

Figure 4.23 tells us further about the behavior of NTP during this interval. The top graph indicates which remote clock the NTP algorithms have selected as the primary synchronization source. The middle graph shows integrated predicted offsets; the same estimated parameters were used to produce this graph and the bottom graph in the previous figure. The bottom graph shows $\texttt{rsadjresid}_{\texttt{integrated}}$, the residuals of rsadj with respect to integrated predicted time, which is $\texttt{rsadj} - \texttt{predrsadj}_{\texttt{integrated}}$. Since predrsadj describes the values the system clock *should have had* with respect to the local oscillator, rsadjresid represents the "error" in how the system clock was steered, assuming that the local oscillator was perfectly described by the model. I consider this to be a view of the behavior of the system clock relative to the local oscillator and its estimated parameters.

The first several days show exemplary behavior. NTP's frequency estimate is stable, and coincides with the value estimated after examining all of the data. It should be noted that NTP's frequency estimate must be produced in real time, and thus may not take into account future data. Actual offsets are small; a few "outlying" observations have magnitudes of several milliseconds. The graph of $\texttt{rsadjresid}_{\texttt{integrated}}$ indicates that the value of the system clock was stable relative to the estimated behavior of the local oscillator; the largest excursion, at midnight on March 2, corresponds to large observed offsets. Values of $\texttt{poffset}_{\texttt{integrated}}$ during this time period are quite consistent also.

The middle portion of the graph, from just after midnight on March 3 until late on March 6, shows significantly different behavior. NTP's frequency estimate is far less stable; excursions of 1 ppm occur many times. Values of NTP's system offset are much larger, ranging from 20 ms to almost −30 ms. Examination of just the top two graphs would suggest that poorer synchronization was obtained during this time period, but there is not a clear picture of what happened to the value of the system clock, and no reasons for this poor behavior are apparent. The graph of $\texttt{rsadjresid}_{\texttt{integrated}}$ indicates that during March 3 the system clock frequently had a value — relative to the estimated behavior of the local oscillator — of approximately 20 ms less than the corresponding values before and after. Examination of $\texttt{poffset}_{\texttt{integrated}}$ during this interval shows that most values of NTP's system offset — when examined relative to the local oscillator — are near the values before and after this period, or are near −15 ms.

The graph of which remote clock is NTP's currently selected synchronization source corresponds closely to the graph of $\texttt{poffset}_{\texttt{integrated}}$. During the first few days and last few days shown in these graphs, the NTP algorithms selected one particular remote clock almost all of the time. During the middle period, other remote clocks were selected, and

[1]The implementation of NTP being run at the time is known as xntp3l.

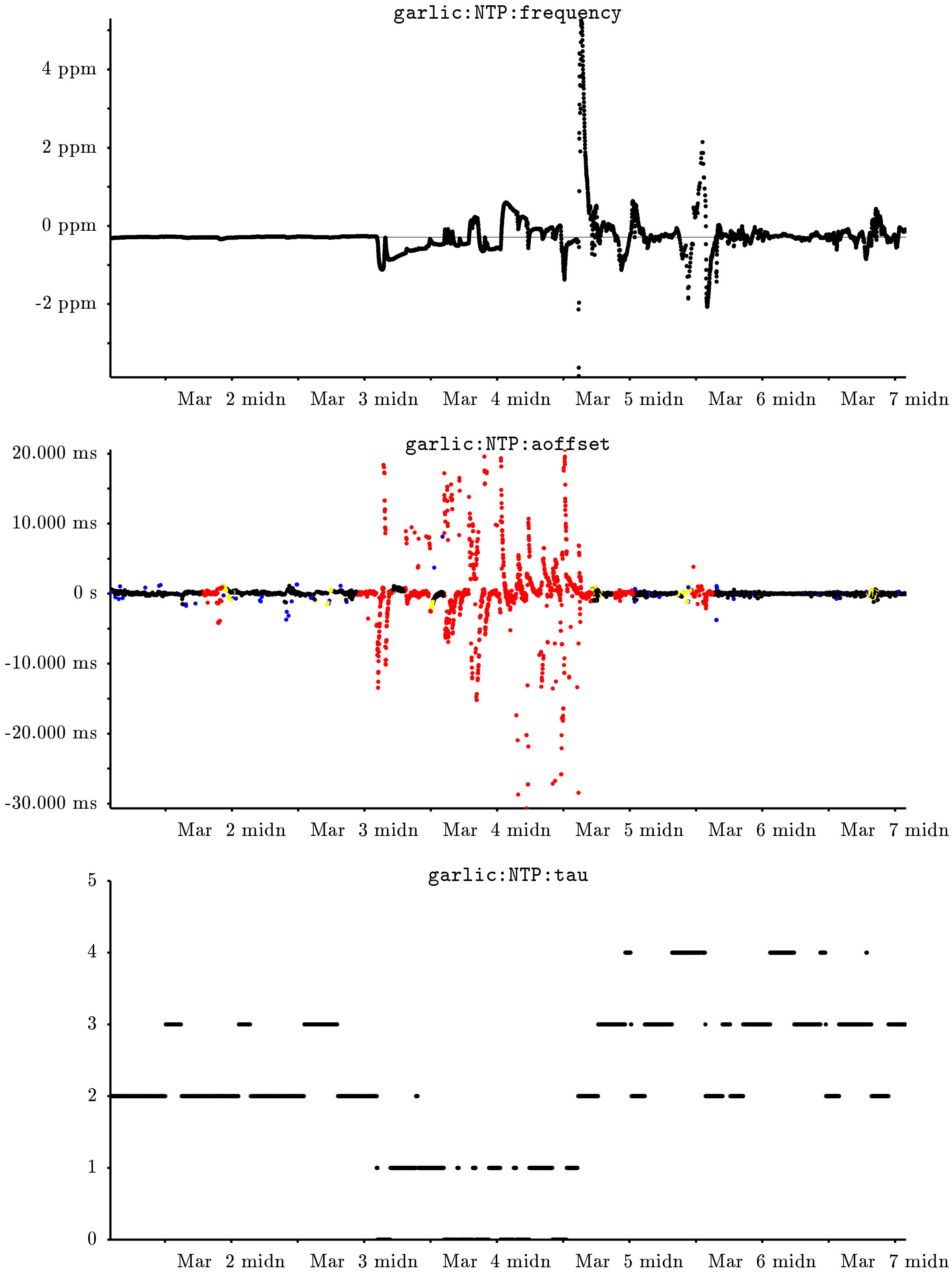



Figure 4.22: NTP's frequency estimates (ppm), top, actual offsets, middle, and indication of loop bandwidth, bottom, vs. time for garlic.

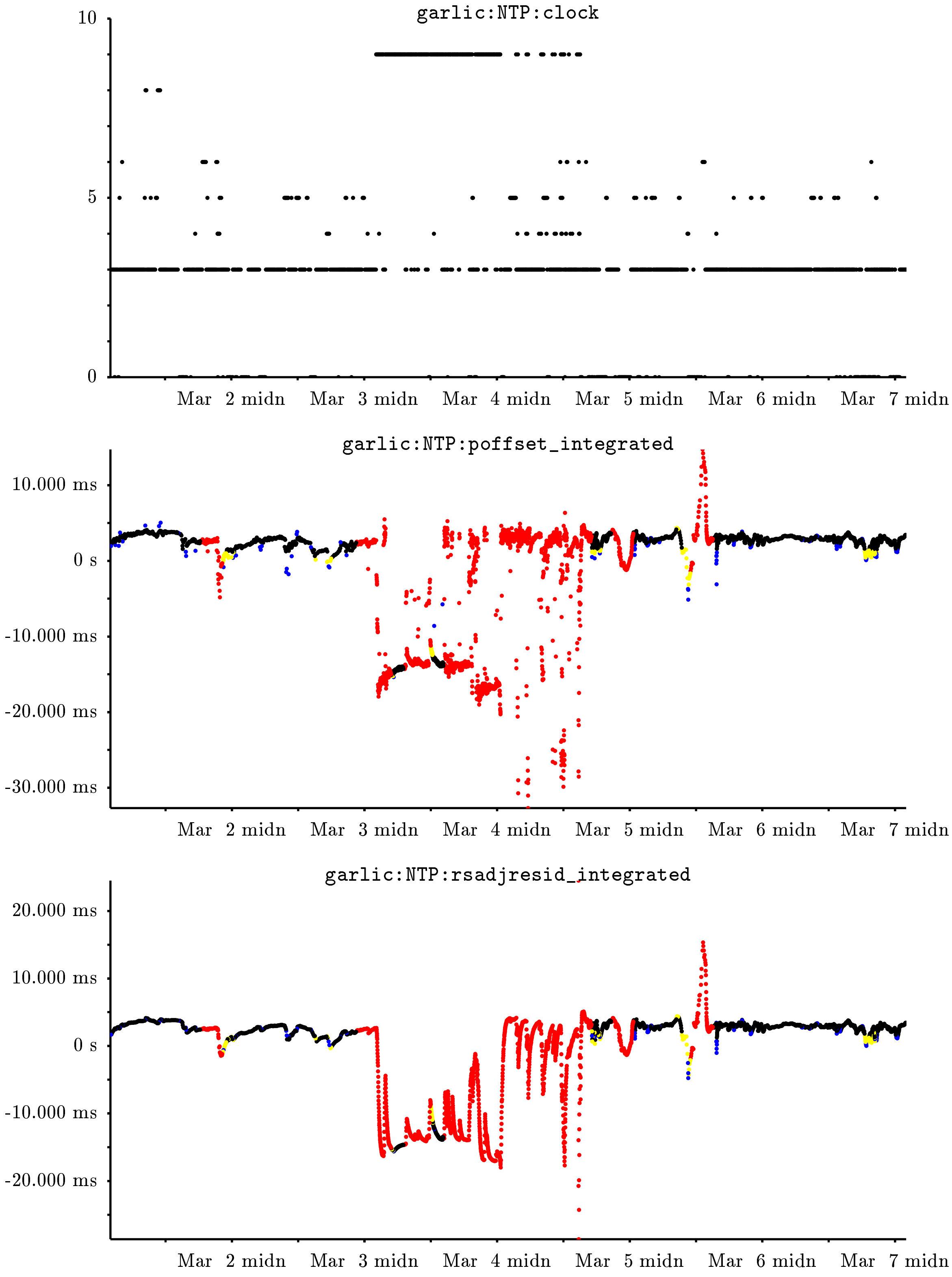



Figure 4.23: NTP's selected remote clock, integrated predicted offsets and integrated residuals of `rsadj` vs. time for garlic.

the algorithms switched among remote clocks frequently. The systematic offsets among the remote clocks are easily seen when examining together the graphs of $\texttt{poffset}_{\texttt{integrated}}$ of the index of the selected remote clock. Each of the several remote clocks selected at various times is more stable than the system offest derived by combining them.

When the value of the system offset changed due to switching remote clocks, the system clock was steered to absorb the change. The frequency estimate was also affected, and later recovered; early on March 3 a negative excursion in frequency was followed by an approximately exponential recovery.

It is interesting to note that the values of `aoffset` can be misleading if not examined very carefully. From examining `rsadjresid` and `poffset` it can be seen that the system clock of garlic experienced significant excursions from some initial phase, and that these excursions corresponded to changes in the selection of remote clock. However, while a few values of `aoffset` are large, many are not; because the NTP algorithms steer the phase of the system clock to match the currently-selected remote clock, the only indication of such an excursion is a few large values at the beginning and end that occur before the phase step is absorbed.

Figures 4.24 and 4.25 show close-up views of the same data for approximately two days near the beginning of the time period of the previous figures. Examining the graphs of `rsadjresid` and NTP's frequency estimate at the same time, striking correlations are evident. NTP's phase-locked loop (PLL) model is responding to nonzero values of `aoffset`; the mechanism changes the value of the frequency estimate and adjusts the phase of the system clock at the same time. Just before midnight on March 2, an excursion in NTP's frequency estimate of 0.04 ppm and in phase of roughly 4 ms is evident; this was caused by a small number of offset observations with values approximately 4 ms different from surrounding observations. Close examination of the data shows that many of these observations had elevated delays; all were marked invalid by the `rsadj` technique.

The graph of NTP's frequency estimate shows that the values tend to be near the estimate produced by the computation of predicted time. Most of the NTP frequency estimates are within 0.01 ppm of the `rsadj` estimate, but several differ by more than 0.05 ppm. While this is not a large frequency error compared to those in Figure 4.22, it is interesting to examine the data in the neighborhood of this excursion. There are several offset observations with negative values just before the excursion; this can be seen in the graphs of both `aoffset` and $\texttt{poffset}_{\texttt{integrated}}$. During the excursion in frequency values of `aoffset` are temporarily elevated; examining $\texttt{poffset}_{\texttt{integrated}}$ shows no such effect. Examining $\texttt{rsadjresid}_{\texttt{integrated}}$ reveals the reason — the system clock had a negative excursion in phase by approximately 4 ms. This is the result of the NTP algorithms adjusting the phase of the system clock to absorb the negative offset observations.

Further inspection of the graph of frequency estimate shows variations of 0.02 ppm from midnight March 2 onward. Values of `aoffset` during this period are almost all less than a millisecond or so. The graph $\texttt{poffset}_{\texttt{integrated}}$, however, displays considerable short-term wander in the average value of offset measurements, almost 3 ms. It should now be considered whether this wander is in the network/remote clock combination, or in the local oscillator. Referring back to the Figure 4.23, it can be seen that the local oscillator was exhibiting stable behavior over the week covered by the graph — a single frequency estimate gave us a good description of many of the offsets. Thus, I conclude that

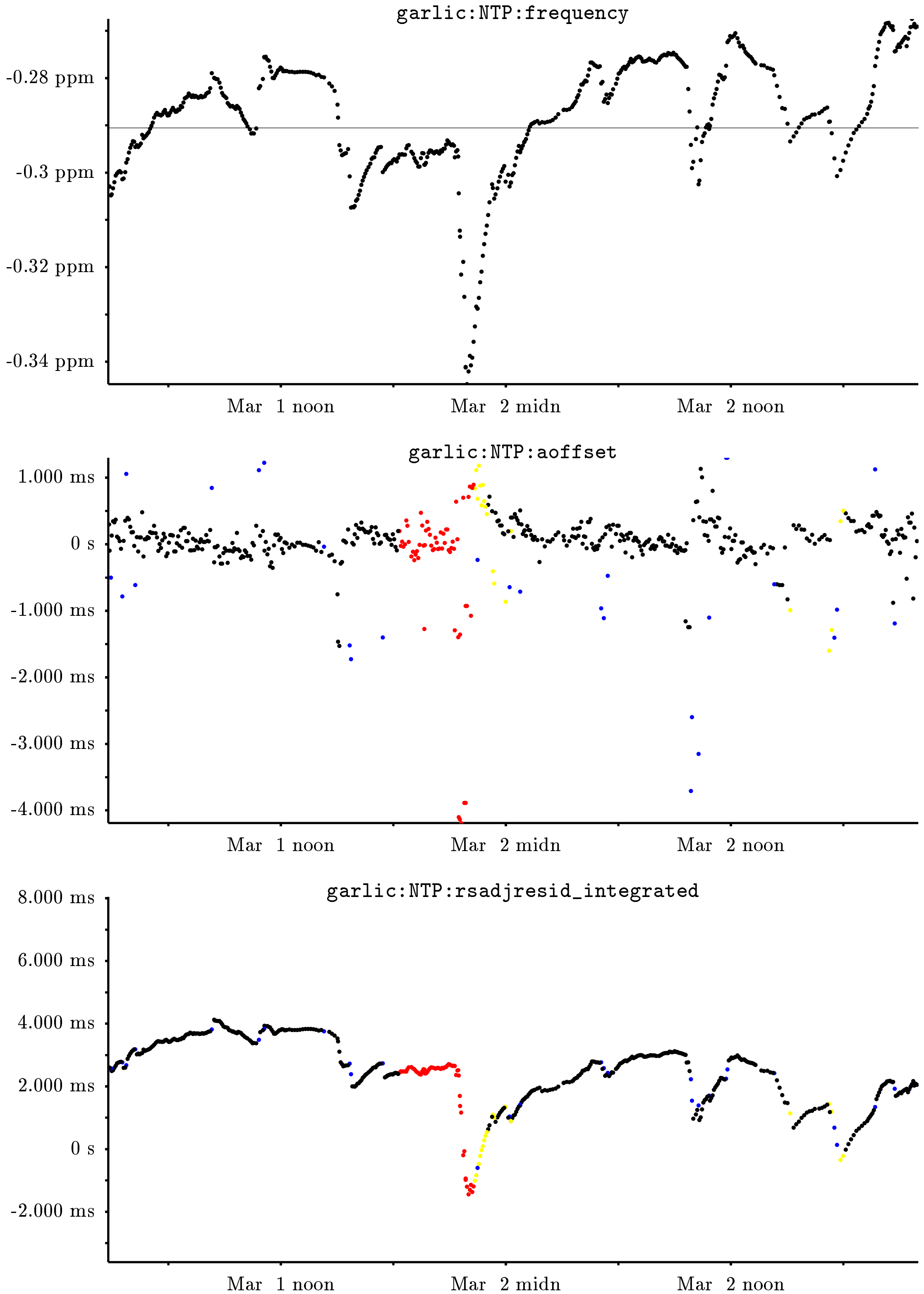



Figure 4.24: NTP's frequency estimates (ppm), top, actual offsets, middle, and `rsadjresid`, bottom, vs. time for garlic.

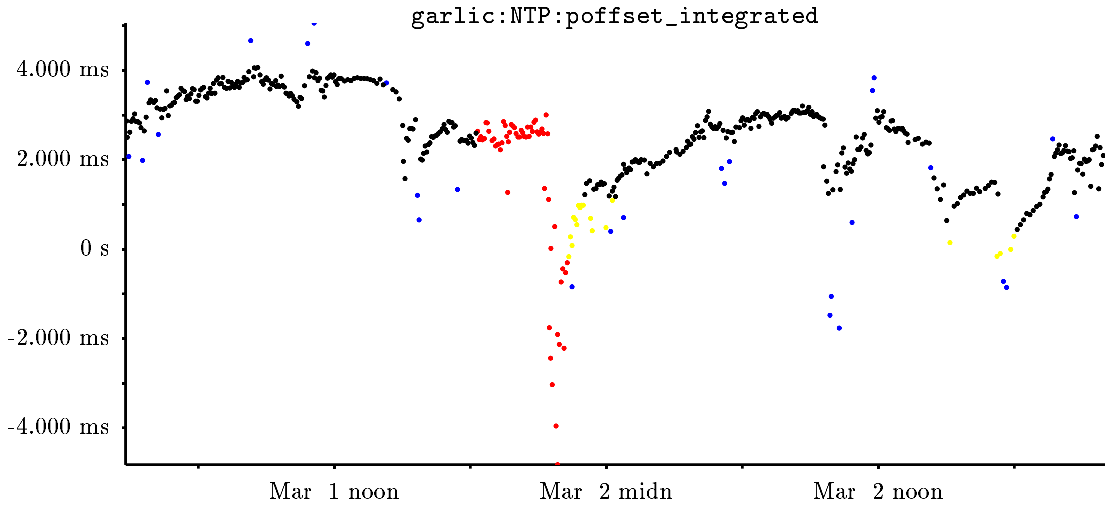



Figure 4.25: NTP's integrated predicted offsets vs. time for garlic.

the wander in `poffset` is caused by the network or the remote clock, but not the local oscillator. The graph of `aoffset`, however, shows none of this structure because the NTP PLL is effectively tracking this wander; this tracking behavior can be clearly seen by noting the correspondence between `rsadjresid` and `poffset` during March 2.

What I have done in the analysis is to separate two processes, given an assumption that the local oscillator is stable. One is the joint behavior of the network and remote clock, and the other is the behavior of the synchronization algorithms. Graphs of `rsadjresid` show the behavior of the synchronization algorithm relative to the local oscillator. Graphs of `poffset` show the behavior of the network and remote clocks relative to the local oscillator. Subtracting `rsadjresid` from `poffset` should therefore show the behavior of the network and remote clocks relative to the system clock. Expanding definitions and then simplifying shows that this is precisely `aoffset`:

$$\texttt{poffset} - \texttt{rsadjresid}$$

$$(\texttt{uoffset} - \texttt{predrsadj}) - (\texttt{rsadj} - \texttt{predrsadj})$$

$$\texttt{uoffset} - \texttt{rsadj}$$

$$\texttt{aoffset}$$

## 4.7 Suggested improvements to NTP local-clock algorithms

In addition to the observations presented previously, I have examined many more graphs of `rsadjresid`, `aoffset`, `poffset`, `delay`, and NTP's frequency estimates. This experience has given me much information about the behavior of NTP's algorithms under the variety of real-world network conditions prevailing in my research environment. From this experience, I am able to make suggestions for incremental improvements to the NTP algorithms to improve their response to some conditions.

When analyzing the behavior of the NTP algorithms with the `rsadj` technique, a frequency estimate derived from data significantly before and after the time period of interest is being used. Also, I have decided, after examining this data, that the local oscillator is indeed stable enough to permit this analysis in many cases. It does appear that both the phase of the system clock and the value of NTP's frequency estimate are being affected more than one would like by a small number of offset samples. It is natural then to ask whether such problems are truly fundamental to the NTP approach to time synchronization, or an artifact of some non-fundamental choice. In either case, it is interesting to consider whether such behavior can be mitigated while retaining the desirable properties of the NTP algorithms. The NTP algorithms do in fact work very well over an extremely wide range of conditions, both in local oscillator stability and in network delay variance. While examining their performance in detail and lamenting small numbers of milliseconds of jitter, the reader should not lose sight of the fact that I can only usually find a few milliseconds about which to complain.

In some sense the unfortunately large response to a small number of large offset observations is fundamental. NTP's model over the short term is a linear phase-locked loop and thus all offset observations that survive the filtering for low delay are weighted equally. The loop gain, computed as a function of a quantity termed "compliance" in the NTP specification, is adjusted over longer time periods. However, the NTP design does not preclude adding mechanisms to identify and discount observations believed to be erroneous.

### 4.7.1 Delay

I observed many circumstances when offset observations with high delays were logged by the modified NTP daemon. Because the modified daemon only logs offset observations that have the lowest delay of the observations currently in NTP's 8-sample filter, these high-delay observations were used by NTP to control the system clock.

The size of the low-delay filter, known as the *clock filter*, cannot simply be greatly increased; the NTP specification document discusses the assumption that the clock-filter delay is small compared to the loop delay, and warns that the loop can become unstable if this is not the case ([Mil92b], p. 93).

I suggest a modification to maintain some sort of estimate of the minimum delay to each remote clock. Possible estimates might be the lowest observed delay over the last 12 hours, the 5th percentile of delays observed over the last 24 hours, or the value that would be the "current low-delay value" used by the low-delay selection algorithm in Section 3.4.2. Then, if the offset observation that is the lowest-delay of the most recent 8 samples is significantly greater than the estimate of minimum delay, that offset observation can be ignored, and NTP's frequency estimate be unchanged for that interval, rather than being affected by the high-delay sample.

The scheme just described is somewhat dangerous, since if the local oscillator is unstable, it is better to track noisy network data than to coast on a bad local oscillator. In order to achieve reasonable behavior in circumstances where delays have increased and the local oscillator is not stable, the offset observation should be used if its magnitude is greater than the amount of error to be expected by the additional delay, i.e., half the additional delay.

### 4.7.2 Systematic offsets

I suggest a modification to estimate systematic offsets among servers, and then correct for them. I saw that switching between remote clocks having different systematic offsets caused significant excursions both in the phase of the system clock and the frequency estimate; it would seem that reducing the effective value of systematic offset would reduce such effects. To implement such a scheme, one could somehow maintain estimates of the relative offsets of the various remote clocks, and subtract these estimates from the observed values when producing the system offset.

Or, the estimates could be produced using the `rsadj` technique, and then configured into the NTP daemon. In this way, long-term trends in systematic offset would be corrected, and any short-term variations would not confound the estimation process, possibly adversely affecting the NTP loop model.

### 4.7.3 Change in adaptive PLL dynamics

The NTP phase-locked loop model changes the loop gain according to an estimate of local oscillator stability, termed compliance, that is intended to approximate a two-sample Allan variance. NTP calculates the mean of the first-order differences of sequential samples ([Mil92b], pp. 94–97). This estimate of stability takes into account the short-term behavior of the network and remote clock as well as the actual stability of the local oscillator. Producing an estimate of stability of just the local oscillator given only noisy measurements is quite difficult. I saw from examining graphs of `poffset` that while some effects were clearly due to changes in frequency of the local oscillator, and in some cases clearly due to behavior of the network and remote clocks, it was not always possible to separate these causes, particularly if only a small amount of data may be examined.

It may be possible to compute a value of compliance that is affected less than the current scheme by a small number of high-value offset observations. I suggest examination of schemes for maintaining the values of the 32 or 64 recent actual offsets in order to compute compliance differently. If the average value of these offsets, or perhaps the median value, is small, the compliance will be small, and the loop bandwidth decreased, causing smaller changes to the phase and frequency estimate for a given observed offset.

### 4.7.4 Setting the loop bandwidth based on the `rsadj` technique

The `rsadj` scheme could be used to determine the loop bandwidth, as it effectively estimates the stability of the local oscillator with respect to the variations in offset caused by network delay variance. The length of the last interval produced by the piecewise computation can be viewed as such an estimate. Perhaps the loop bandwidth could be controlled by some function of this interval length. If the piecewise computation produces a last interval of 2 days when configured with a combining limit of $4E_n$, perhaps $2/4 = 0.5$ days should be considered a reasonable time period, and the loop bandwidth set to $\frac{1}{12\mathrm{h}}$. Such a scheme has potential drawbacks; if the characteristics of the local oscillator change, the loop dynamics would not be updated until the piecewise computation is next done. Investigation into the possible use of such a scheme should be done keeping such issues in mind.

## 4.8 GPS-based verification of the `rsadj` technique

I used the GPS receiver attached to `ipa` as a trusted reference — a source of *true time* — in order to verify the results obtained with the `rsadj` measurement technique. This provided a way to verify that assumptions made about the local oscillator were valid. It also provided a way to observe the actual value of the systematic offsets already seen among the various remote clocks. With a source of true time, it can be said which of them is correct, rather than simply ahead of or behind each other.

### 4.8.1 Quality of GPS time

Before using a particular GPS receiver as a source of true time, it should be determined that the time provided by such a receiver is accurate. The GPS receiver used in this research outputs on-time pulses at 1 pulse per second; the rising edge of the pulses are claimed to be within 55 ns of UTC by Magellan, the manufacturer, when Selective Availability (SA) is not enabled. SA is intentional degradation of the positioning capability of the GPS system by the Department of Defense to restrict the access of non-military users to precise positioning. It has generally been enabled during this work. Even with SA enabled, GPS is specified to provide two-dimensional rms errors of 100 m or less 95% of the time. Since an error of 100 m is equivalent to a timing error of 300 ns, it can certainly be expected that the vast majority of GPS-derived timing pulses used in this part are accurate to within 1 $\mu$s. Because I plot GPS-derived data for inspection, but don't otherwise process it, occasional erroneous samples are not problematic in the use of GPS-derived timing information for validating the measurement technique.

### 4.8.2 Values of the system clock with respect to GPS

The GPS receiver has a precise timing output referred to as an *on-time pulse*. This signal undergoes a rising edge at the beginning of every second. By observing the values of the system clock when the rising edge occurs, the error in the value of the system clock can be measured.

Consider the set of values of the system clock at the moments of the GPS-derived on-time signal. For each such value, construct an ordered pair with the nearest integer value as its first component, and the difference between that integer value and the system clock as the second component. I term the set of such pairs `capture`. Thus, `capture` is a set of (`atime`, $c$) values, where $c$ is the "fractional" part of the system clock, such that $-0.5$ s $< c \leq 0.5$ s.

It is important to note that `capture` $\neq$ `aoffset`, but that `capture` $=$ $-$`aoffset`. If an offset observation to the GPS receiver had been made at the instant of the rising edge, the offset obtained would be the negative of the fractional value of the system clock, as the fractional value of time on the GPS receiver would be zero. For example, assume that the value of the system clock was 764862293.987250 s at the rising edge of the on-time signal. Because the value recorded was just less than an integral number of seconds, it indicates that the system clock was slow compared to the GPS receiver, and this is a positive offset. Assuming that the second indicated by the rising edge is in fact the nearest second to the system clock value the offset would be 764862294 s – 764862293.987250 s, or 0.012750 s I thus consider this to be an `aoffset` value of $(764862294\text{ s}, 0.012750\text{ s})$. A system clock value of 764862294.002491 s, on the other hand, would be considered an aoffset value of (764862294 s, 0.002491 s); in this case the system clock is 2491 $\mu$s ahead of GPS time. Since capture can be written as

$$\texttt{capture} = -\texttt{aoffset}_{\texttt{GPS}} = -(\texttt{atime}_{\texttt{GPS}} - \texttt{atime})$$

if GPS time is accepted as true time, one can write

$$\texttt{capture} = \texttt{atime} - \texttt{ttime}$$

The values of capture are actually obtained by connecting the on-time signal from the GPS receiver to the capture input of the custom-built clock board installed in ipa. When a rising edge of the on-time signal occurs, the clock board copies the low-order 32 bits of the 64-bit counter into the capture register. When the counter is read, the capture register is checked, and, if it contains a new value, the value of the system clock corresponding to the value is computed. Only every 60th such value is logged in order to reduce the volume of data that must be stored to a manageable level. Log messages from the GPS receiver indicating the "time-quality figure of merit" are reduced and logged. When creating plots of capture data or data derived from it, only samples from time periods when the GPS receiver indicated that it was producing valid timing pulses are used.

### 4.8.3 Transferring GPS time across an ethernet

Due to equipment constraints, the GPS receiver used in this research is not attached to the computer serving as the oscillator testbed. Thus, I introduce a mechanism to transfer precise GPS time from ipa, the computer attached to the GPS receiver to garlic, the computer to which the various oscillators were attached. From the values of capture on ipa, I produce *synthetic capture* information for garlic, which I term $\texttt{capture}_{\texttt{garlic}}$.

In addition to recording values of $\texttt{capture}_{\texttt{ipa}}$, ipa is configured to make offset observations of garlic, which I shall refer to as $\texttt{aoffset}_{\texttt{garlic-ipa}}$. I wish to produce values of $\texttt{capture}_{\texttt{garlic}}$, or values that would have been obtained had the GPS on-time signal been brought to suitable hardware garlic.

From $\texttt{capture}_{\texttt{ipa}}$, the capture data on ipa, and $\texttt{aoffset}_{\texttt{garlic-ipa}}$, the offset measurements of garlic made from ipa, I can compute $\texttt{capture}_{\texttt{garlic}}$, synthetic capture data for garlic. Let

$$\texttt{capture}_{\texttt{garlic}} = \texttt{capture}_{\texttt{ipa}} + \texttt{aoffset}_{\texttt{garlic-ipa}}$$

Expanding the definition of capture and aoffset,

$$\texttt{capture}_{\texttt{garlic}} = \texttt{atime}_{\texttt{ipa}} - \texttt{ttime} + \texttt{atime}_{\texttt{garlic}} - \texttt{atime}_{\texttt{ipa}} + \texttt{noise}$$

Simplifying,

$$\texttt{capture}_{\texttt{garlic}} = \texttt{atime}_{\texttt{garlic}} - \texttt{ttime} + \texttt{noise}$$

An example may be helpful in order to provide insight into the construction of synthetic capture data. Recall that a positive value of $\texttt{aoffset}_{\texttt{garlic-ipa}}$ indicates that garlic's clock is ahead of ipa's clock. Recall that a positive value of $\texttt{capture}_{\texttt{ipa}}$ indicates that ipa's clock is ahead of true time. For example, if at a particular time $\texttt{capture}_{\texttt{ipa}}$ is 5 ms, $\texttt{aoffset}_{\texttt{garlic-ipa}}$ is 2 ms, then $\texttt{capture}_{\texttt{garlic}}$ is 7 ms. This is sensible; ipa is 5 ms fast, and garlic is 2 ms ahead of ipa, and therefore garlic is 7 ms fast.

### 4.8.4 Measuring an oscillator with GPS

The value of capture tells us the relationship between a computer's actual time and true time. I can combine this with rsadjresid to obtain the relationship between the local oscillator and true time.

$$\texttt{oscillator} = \texttt{capture} - \texttt{rsadjresid}$$

Expanding,

$$\texttt{oscillator} = (\texttt{atime} - \texttt{ttime} + \texttt{noise}) - (\texttt{rsadj} - \texttt{predrsadj})$$

$$\texttt{oscillator} = ((\texttt{utime} + \texttt{rsadj}) - \texttt{ttime} + \texttt{noise}) - (\texttt{rsadj} - \texttt{predrsadj})$$

$$\texttt{oscillator} = (\texttt{utime} + \texttt{predrsadj}) - \texttt{ttime} + \texttt{noise}$$

$$\texttt{oscillator} = \texttt{ptime} - \texttt{ttime} + \texttt{noise}$$

The result is the difference between predicted time and true time, corrupted by noise. If the local oscillator were perfectly modeled by the estimated parameters and the offset measurements were all correct, ptime would be equal to ttime. However, this is not the case due to variations in frequency error of the local oscillator, and because the estimated parameters will not be exactly correct. In general, then the value of oscillator will have a systematic phase and frequency error, as well as less easily characterized variations.

### 4.8.5 Offsets relative to GPS time

Given values of capture, values of goffset, the offsets that would have been observed had the local clock been synchronized to GPS time, can be computed. These is very close to toffset, the offsets that would have occurred had the local clock had the correct time.

To obtain the GPS-referenced offsets $\texttt{goffset}_{\texttt{a-garlic}}$ for some remote clock a, one adds $\texttt{capture}_{\texttt{garlic}}$ and $\texttt{aoffset}_{\texttt{a-garlic}}$:

$$\texttt{goffset}_{\texttt{a-garlic}} = \texttt{capture}_{\texttt{garlic}} + \texttt{aoffset}_{\texttt{a-garlic}}$$

Expanding,

$$\texttt{goffset}_{\texttt{a-garlic}} = (\texttt{atime}_{\texttt{garlic}} - \texttt{ttime} + \texttt{noise}) + (\texttt{atime}_{\texttt{a}} - \texttt{atime}_{\texttt{garlic}} + \texttt{noise})$$

$$\texttt{goffset}_{\texttt{a-garlic}} = \texttt{atime}_{\texttt{a}} - \texttt{ttime} + \texttt{noise} + \texttt{noise}$$

Note that in this case the two terms written noise refer to different offset measurements, and thus do not cancel or add.

## 4.9 Observations with GPS-based verification

Using the GPS receiver on ipa, I can obtain much information about timekeeping performance. Also, I can verify that much of the information can be obtained via the rsadj technique without the use of GPS.

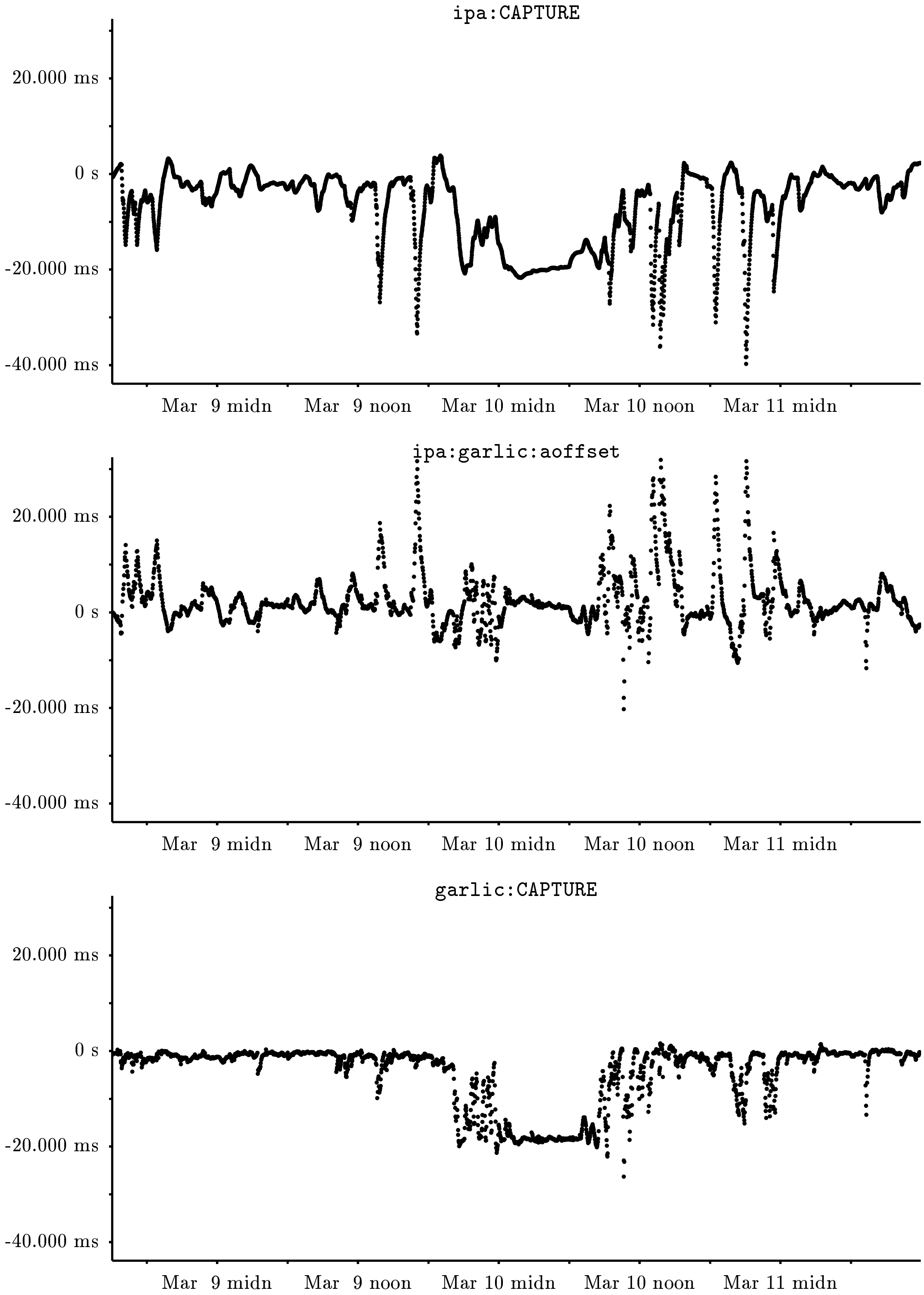



Figure 4.26: Capture data for ipa, actual offsets of garlic as observed by ipa, and synthetic capture data for garlic.

The top graph in Figure 4.26 shows $\texttt{capture}_{\texttt{ipa}}$. Note that the values are locally very smooth, and thus there is no reason to believe that it is problematic to record only every 60th value as I have done. The second graph shows that there is some noise in the offset measurement between the two machines, but that most observations encounter only small errors. The third graph is of $\texttt{capture}_{\texttt{garlic}}$; this shows the behavior of $\texttt{atime}_{\texttt{garlic}}$ with respect to ttime. The effects of switching to a difference reference clock can be clearly seen in the center of the graph.

Figure 4.27 shows capture, integrated residuals of rsadj, and oscillator for garlic. The top two plots are strongly similar. This should not be surprising; capture shows the view of the system clock with respect to GPS time, and rsadjresid shows it with respect to the estimated behavior of the local oscillator. Subtracting these yields oscillator, which shows the behavior of the local oscillator and its estimated parameters with respect to GPS time, corrupted by the ethernet noise from ipa to garlic.

One can compare plots of goffset and poffset, and see that they are very similar. Figure 4.28 shows poffset and goffset to apple.com from garlic. GPS data are missing for part of this interval due to problems with ipa. The values are generally similar in character, but one can see that apple.com appears to have a systematic offset relative to GPS time. While this is visible in the plot of goffset — which does not involve any estimation — it has been estimated and corrected for in the plot of poffset.

One might consider subtracting poffset from goffset, but since

$$\texttt{goffset} = \texttt{aoffset} + \texttt{capture}$$

and

$$\texttt{poffset} = \texttt{aoffset} + \texttt{rsadjresid}$$

it can be seen that

$$\texttt{goffset} - \texttt{poffset} = \texttt{capture} - \texttt{rsadjresid} = \texttt{oscillator}$$

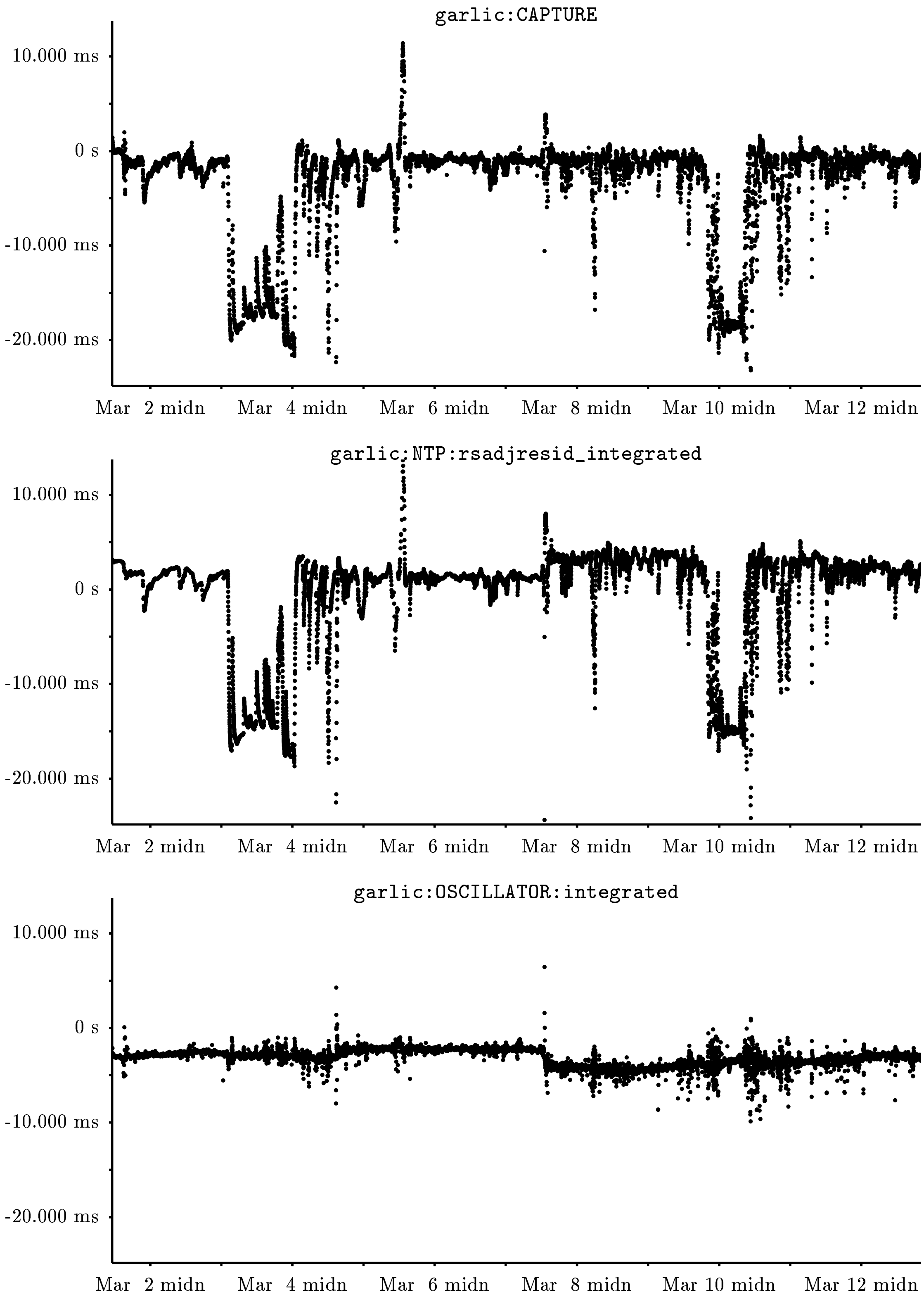



Figure 4.27: Capture data, residuals of `rsadj`, and `oscillator` for garlic.

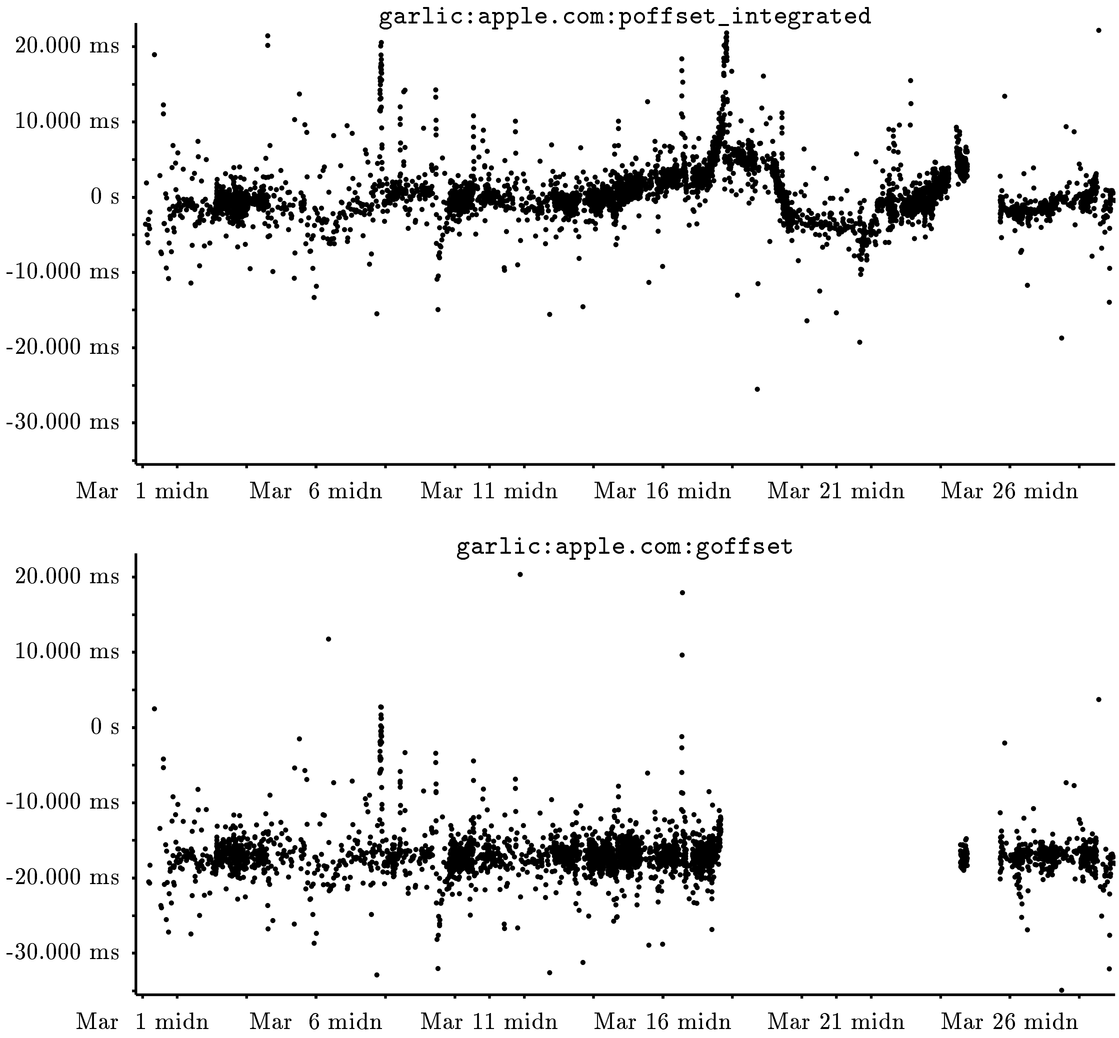



Figure 4.28: Predicted offsets and `goffset` of apple.com as observed by ginger. GPS data are missing for part of this interval due to problems with ipa.

# Chapter 5
# Integrated Timekeeping

In this chapter I present a new scheme to synchronize clocks over packet networks, called *integrated timekeeping.* The scheme assumes that offset measurements to remote clocks over packet networks are made periodically. The scheme is based on the `rsadj` measurement technique; it predicts the future behavior of the local oscillator from the estimated parameters of local oscillator. I term this approach *integrated timekeeping* because the scheme takes as input all available data, and, most importantly, views the local oscillator as a device that may be characterized; it is both the measuring instrument and an object of study.

The central idea of integrated timekeeping is to steer the system clock according to estimates of the local oscillator parameters, rather than in accordance with recent offset observations. New estimates are produced at a rate sufficient to ensure that the errors due to unmodeled changes in the local oscillator do not become significant between updates. Integrated timekeeping builds on the techniques previously developed to produce accurate estimates of the parameters of the local oscillator. Further refinement of the techniques are necessary. It is not enough that estimated parameters be accurate; they must also change very little as new observations are made. A phase estimate of 2.4 ms may be little different than one of 2.1 ms, but a change in phase of 0.3 ms every time new estimates are computed is undesirable. I describe refinements to the integrated computation of predicted time to avoid such small changes It is also necessary to avoid applying statistical characterizations, such as those of delay, derived from extremely old data to recent data. I describe refinements to the integrated computation so that the results are not affected by data from arbitrarily long ago.

After describing the integrated timekeeping scheme, I present data obtained via simulation showing the performance of integrated timekeeping relative to that of NTP, given a local oscillator stable to a few parts in $10^8$. By simulating integrated timekeeping, I can directly compare the performance of the two synchronization algorithms when presented with the same data. I show that integrated timekeeping achieved values of the system clock stable to better than 2 ms, while NTP achieved values stable to only 20 ms. I also present data showing the behavior of an implementation of integrated timekeeping. In particular, I show that a stability of several milliseconds was achieved by the integrated scheme when using only two remote clocks in Europe and one in Australia. In contrast, NTP achieved system clock stabilities of no better than 15 ms with these same clocks a few days later.

I present data showing the effects of disabling parts of the computation of integrated predicted time on the performance of the integrated timekeeping scheme. The adverse effects of these changes can be clearly seen from the data presented.

While I show that with local oscillators stable to a few parts in $10^8$, integrated timekeeping works well, it is also true that integrated timekeeping performs quite poorly given a local oscillator with a stability of only a few parts in $10^6$. I present data showing this poor performance.

The integrated timekeeping scheme can be used to replace the NTP local-clock algorithms. However, many processing steps described by the NTP specification would continue, such as deciding how often to attempt to make offset observations. The clock selection and combining steps are still done, but their results are not used to control the system clock. Instead, they are used as a sanity check, ensuring that the integrated timekeeping scheme does not cause the system clock to be grossly incorrect. Grossly incorrect values of predicted time are unlikely, but could result under anomalous conditions in which the normally valid assumptions made by the computation of predicted time do not hold.

I discuss issues related to cascading multiple computers, each running the integrated timekeeping scheme, and present data showing reasonable performance in the case of two such cascaded computers. I am unable to offer a proof that the scheme is stable given long chains of cascaded computers each running it, but I argue that as long as each computer is able to maintain accurate values of its system clock, the next computer will have accurate data from which to characterize its local oscillator.

I believe that the integrated timekeeping scheme is more successful than NTP in synchronizing clocks given stable local oscillators for several reasons. At the end of the chapter, I discuss these reasons and the relevant fundamental differences between NTP and integrated timekeeping.

## 5.1 Description of the integrated timekeeping scheme

In Chapter 3, the goal was to compute *predicted time* for the purpose of analyzing synchronization performance. Values of `predrsadj` were computed for various intervals that minimized the values of `poffset`. The intervals were those such that the behavior of the local oscillator could be reasonably characterized by a single set of parameters for each interval. The integrated timekeeping scheme consists of periodically computing `predrsadj` based on past observations. Previously, several different kinds of predicted time or `predrsadj` were computed: piecewise predicted time for NTP data, piecewise predicted time for each set of raw data, and integrated predicted time, based on raw data from selected remote clocks.

Using the `rsadj` synchronization measurement technique involved computing `predrsadj` and considering `poffset`, the offset values that would have been observed if `rsadj` *had been* `predrsadj`, and `rsadjresid`, the difference between `rsadj` and `predrsadj`. The integrated timekeeping scheme specifies that `rsadj` be set equal to `predrsadj`. This is a straightforward operation — since

$$\mathtt{rsadj} = \mathtt{atime} - \mathtt{utime}$$

the system clock is simply commanded to behave such that

$$\mathtt{atime} = \mathtt{utime} + \mathtt{predrsadj}$$

Assume for a moment that the local oscillator can be perfectly modeled by the three parameters of the local oscillator model, and that these parameters can be correctly estimated. Then, one could simply compute `predrsadj` once and use it to control the system clock for all future time, and the system clock would have exactly the correct time — `atime` would equal `ttime` — forever into the future. Of course, these assumptions are not valid. As shown in the previous chapter, typical local oscillators do exhibit deviations from the model. And, while the computation of predicted time attempts to estimate parameters as

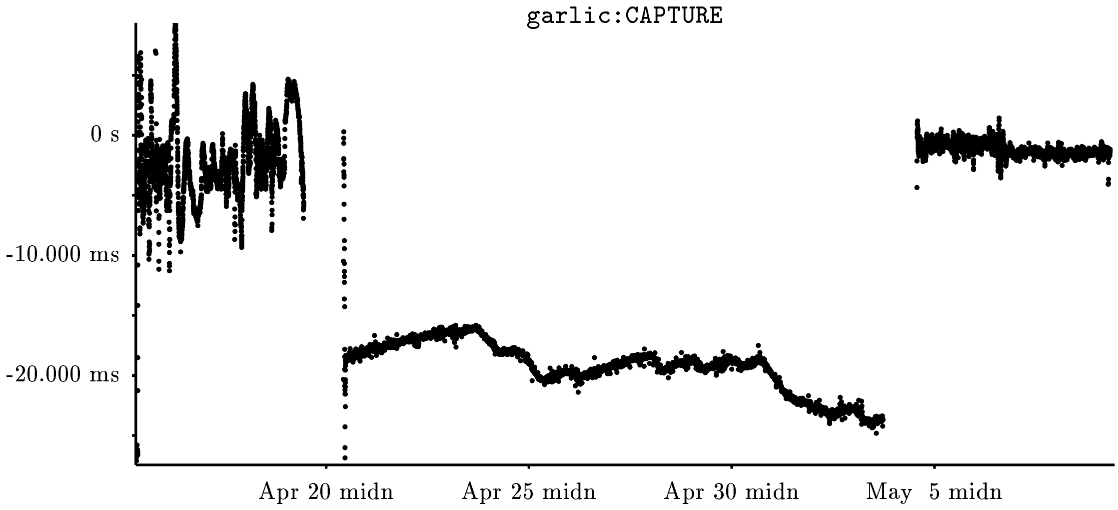



Figure 5.1: `capture` vs. time for garlic, showing the effects of the system clock being commanded to follow a single value of `predrsadj`. While the value of the system clock is locally stable, it begins to wander, eventually becoming excessive. During the left and right regions the system clock was controlled by NTP.

well as possible, the estimates will never be exactly correct, as there are sources of error within the system.

Figure 5.1 shows the results of attempting to predict time based on one estimate. On April 20, near the left side of the graph, estimates were computed, and the NTP daemon running on garlic was instructed to compute `atime` based on those estimates. The resulting timekeeping performance — shown relative to GPS-derived time — was reasonable over a short period of time, but then the effects of frequency changes in the local oscillator became increasingly evident. The phase difference from NTP is due to a different weighting of remote clocks. Consider for a moment the effect of an error of 0.01 ppm in a frequency error estimate. Multiplying by one day, or 86400 s, gives 864 $\mu$s, or close to 1 ms. Thus, keeping the accumulated timekeeping error due to misestimation of frequency error to less than 1 ms per day requires estimating frequency error to within 0.01 ppm. It also requires that unmodeled frequency changes be less than 0.01 ppm.

Rather than attempting to produce one set of estimates, and using that for all future times, the integrated timekeeping scheme produces an estimate to use until a more recent — and thus presumably better — estimate is available. While the system clock is being controlled by one set of estimates, the timekeeping daemon continues to record the results of offset observations. Because the `rsadj` scheme is able to process observations independently of the value the system clock, it may use these observations in exactly the same manner as before, and thus integrated `predrsadj` may be computed again, including the new data.

I have already discussed and rejected one extreme choice of how to compute `predrsadj`, that of computing it only once. The other extreme, computing `predrsadj` every time a new offset observation becomes available, is unattractive because it is very computationally intensive. Thus, I consider the practice of computing a new value of `predrsadj` at some interval, such as every hour. Computing `predrsadj` more frequently requires more resources. Computing `predrsadj` less frequently will result in greater accumulated errors due to misestimation of frequency error and changes in frequency in the local oscillator.

A more subtle issue is that computing `predrsadj` more frequently will result in more frequent *changes* in the value of time returned by the system clock. It cannot be reasonably said that one estimate is better than another if they only differ slightly in phase, because of the impossibility of determining offsets over a packet network. However, using one or the other is preferable to switching between them every few seconds, as the choice of a single estimate is much more stable in the short term. In many respects this is similar to the problem of NTP choosing different remote clocks in computing the system offset. I deal with this problem in two ways. One is to compute `predrsadj` at an interval chosen to balance the effects of frequency changes and short-term instability induced by differing estimates. The other is to compute `predrsadj` in such a way that successive estimates are similar in phase.

## 5.2 Computation of predicted time for integrated timekeeping

This section explains how computation of predicted time for integrated timekeeping differs from the computation for analysis presented in Chapter 3. Some changes involve attempting to have the estimates reflect the recent past — and thus the future — more accurately, while being less concerned with obtaining a view of performance in the past. Others involve "stabilizing" the computation, so that as predicted time is computed on an augmented set of observations, reasonably consistent values of predicted time are produced.

### 5.2.1 Lower combining limit

In integrated timekeeping, the goal is that the value of `ptime` over the next estimation period — typically a few hours — be close to `ttime`. Secondarily, as much data as possible should be used in order to reduce the errors due to network delay variance. In contrast, when performing analysis, the goal was that `ptime` be close to `ttime` over the entire period of the analysis, and that fewer rather than more intervals be used. Thus, I choose a smaller combining limit than for the analysis computation.

In Section 3.4.7, I chose for the analysis computation's combining limit a value of 4 times the apparent error caused by network delay variance, or $4E_n$. I choose $2.5E_n$ for use in the integrated timekeeping scheme, a value empirically observed to work well. This value is chosen in order to reduce the amount of tolerated error due to frequency changes, but still allow averaging over errors due to network delay variance. It is only slightly greater than the value of $2E_n$ which would result in equal error contributions from delay variance and frequency changes — if the error sources were uniform over time. The value of $2.5E_n$, however, mitigates the problem of adjacent segments of rms error close to $2E_n$ not being combined. In the case of two segments of maximum error, $2E_n$, the segments will be combined if the computed measure of similarity is less than $2.5E_n$. One can consider that the amount of error due to unmodeled frequency change is then one-fourth of the error due to network delay variance. In the average case, that of segments with rms errors of $1E_n$, the additional error due to frequency changes will be $1.5E_n$.

### 5.2.2 Discarding of very old data

In the analysis computation, all available data were used. In general, this produced the desired outcome — a set of intervals and corresponding estimates describing the local oscillator. However, even very old data affected the output of computation for the recent past, and this is undesirable when predicting the future, as some old data may be sufficiently old that it is harmful rather than helpful. Thus, I added a mechanism to control how much old data is used in the computation.

Even very old data could affect the outcome of the estimates for the most recent time period in several ways. When marking invalid offset observations with high delay, the computation used statistical information over all observations. If the oldest data had a different underlying delay distribution, a different set of observations might be marked invalid. When determining $E_n$, used for both marking invalid high-variance segments and for determining the combining limit, the computation used information from segments containing all previous observations. If something changed in the local oscillator, network, or remote clocks, statistical information derived from old data might no longer be valid. For example, if the last interval produced in the piecewise computation of predicted time spans the last 7 days, two year old data are not particularly helpful in estimating the oscillator frequency, and there is no reason to believe it is relevant.

I address this issue by deciding, for each remote clock, how much data to use before any other processing steps. Because integrated predicted time and therefore piecewise predicted time is repeatedly computed, a particular computation has available the results of the last computation. The computation of integrated time should use enough data so that an accurate statistical characterization of the recent past can be obtained, but not so much that old and possibly irrelevant data are used. I chose to use data covering twice the time period of the last interval of the previous computation, subject to a configured maximum and minimum additional amount. The chosen value for the minimum additional amount is 4 days in order to ensure that there are enough segments over which to compute the median rms error for the determination of $E_n$, the rms error due to average network delay variance. If the previous interval were only 4 hours, 8 hours of data would likely not suffice to provide a sufficient basis to decide which segments are higher than acceptable variance. The chosen value for the maximum additional amount is 7 days; this limits the time period over which data from before change in the local oscillator or network conditions can affect the computation.

It is quite possible that only one value would be better, in effect examining $n$ days of previous data regardless of the length of the previous interval. I claim that the exact choice is not critical — the important points are that enough previous data be examined to allow accurate statistical characterization of delay and median rms error, and that not so much be examined to cause an undue reliance on old data.

### 5.2.3 Adding hysteresis to piecewise segmenting

As piecewise predicted time is calculated, the new data may cause the last interval produced by the piecewise computation — the one used for the computation of integrated predicted time — to have a different beginning time. While the end of the interval obviously increases each time, the beginning may remain the same, move backward or forward in time as the next estimation is done. It may move forward if the new data causes the median rms error of the short segments to decline, thereby reducing the combining limit. Conversely, it may move backward if the new data causes the median rms error to increase.

This dynamic determination of the interval for piecewise predicted time is in general a necessary part of the scheme — it is necessary in order to achieve a balance between errors due to network delay variance and unmodeled behavior of the local oscillator. However, fluctuation between several beginning times is harmful, as estimates using different periods will generally have slightly different phase and frequency error estimates. While it is difficult to say which of the results — from the shorter or the longer fit — is better, switching frequently between them is clearly bad, as it introduces abrupt changes in phase.

In order to avoid this problem, the integrated timekeeping scheme retains state across prediction periods, recording the time period of the last interval from the previous computation of piecewise predicted time for each remote clock. A scheme to prevent oscillation of the beginning time is introduced; the beginning time may become later in time, and may only become earlier if the earlier beginning time persists over several computations.

I describe the mechanism of computing the beginning time to be used mathematically. Let the various instances of the computation of predicted time be numbered sequentially from 0. Let the beginning time of the last interval computed for piecewise predicted time be termed $b_i$ for the $i$th computation. For each computation $i$, let $a_i$ represent the beginning time at which data are actually used in the integrated computation. Let $a_0 = b_0$, so that the actual beginning time computed will be used on the first iteration. If $b_i \geq a_{i-1}$, then $a_i = b_i$. In other words, if the interval just computed has the same or later beginning time, that just-computed beginning time is used. If $b_i < a_{i-1}$, then in general $a_i = a_{i-1}$. In other words, if the interval just computed has an earlier beginning time, the previous time is used. However, if $b_i = b_{i-1} = \cdots = b_{i-k}$, then $a_i = b_i$. That is, if the same beginning time has been computed $k$ times in a row, that beginning time is used, even if it is earlier than the previously used beginning time. In the implementation of the computation created for the thesis research, $k$ is 5.

### 5.2.4 Changes in the values of systematic offsets

In the computation of integrated predicted time, systematic offsets of the various remote clocks are estimated. As remote clocks are included or not included in the integrated computation, or even as the intervals over which they are examined change, the average value of phase will change. Therefore, the computed values of systematic offsets will change, as they are relative to the average value.

For example, assume remote clock $A$ is in fact 4 ms ahead of remote clock $B$. Assume that in a particular computation of integrated time the effective weight of $B$ is three times that of $A$, perhaps due to the last interval of $A$ being roughly three times longer. In this case the systematic offsets might be 3 ms for $A$ and $-1$ ms for $B$; $A$ is in this case 4 ms ahead of $B$, and the weighted sum is zero. Now, assume that the next time the computation is performed the last interval of $A$ is the same size as of $B$, and the two remote clocks have equal weight. This time the estimated systematic offsets will be 2 ms for $A$ and $-2$ ms for $B$. The estimates still differ by 4 ms, and the weighted sum is still zero. However, the notion of zero has effectively shifted by 1 ms. The integrated computation attempts

```
Updating stored systematic offsets:
6 offsets, average stored offset -0.189 ms, update -0.044 ms
Clock  0 offset  9.469 ms sys  0.068 ms update  0.016 ms new_offset  9.485 ms
Clock  1 offset -6.309 ms sys -0.106 ms update -0.025 ms new_offset -6.335 ms
Clock  2 offset -7.047 ms sys -0.001 ms update -0.000 ms new_offset -7.047 ms
Clock  3 offset  5.277 ms sys  0.045 ms update  0.011 ms new_offset  5.288 ms
Clock  4 offset 11.210 ms sys -0.107 ms update -0.025 ms new_offset 11.184 ms
```

Figure 5.2: Sample output from the computation of integrated predicted time for use in the integrated timekeeping scheme. The second line indicates that the average value of systematic offset stored for 6 clocks is $-0.189$ ms, and that $-0.044$ ms was added to each stored offset to move the average closer to zero. The rest of the lines give information about remote clocks used in this computation of integrated predicted time. The numeric columns are clock number, stored estimated systematic offset, the estimated offset relative to the stored value, the amount to be added to the stored value, and the new stored value.

to estimate predicted time that is a weighted sum of the various remote clocks, and the estimated phase is thus effectively a weighted sum of the phases that would be estimated for each clock. Changes in weight caused by changes in the intervals of remote clocks thus cause phase changes in the resulting offset if the remote clocks have different phases.

While it is not possible to say which remote clock has the correct phase, it is undesirable to switch which value of phase to track every time integrated time is computed. Thus the long-term systematic offset of each remote clock is estimated, and deviations from that saved offset are estimated, rather than the overall offset. The long-term estimate is computed by exponential averaging; in the current scheme, the configurable constant for the time constant is set so that 80% of a perceived step change in offset will be absorbed in one day. Because this scheme could result in arbitrary drift of the average value of estimated systematic offsets, the average value of all the estimated offsets is is driven to zero by exponential decay with the same time constant. The value of 80%/day was chosen somewhat arbitrarily; the average is slowly-moving enough to avoid responding greatly to any one computed value when the computation is performed every 4 hours, but not so slow as to cause the estimated offset differences to be often large, and thus induce phase changes on the resulting computed predicted time. Other choices of averaging schemes may be more appropriate, and are beyond the scope of this thesis; this scheme is presented to illustrate the problem and show that a reasonable solution exists.

This averaging scheme estimates a long-term value of systematic offset for each remote clock, and then corrects for this systematic offset before estimating the remaining differences among remote clocks. In an ideal world, the actual systematic offset between remote clocks would be zero. This would require, however, that all systematic biases in remote clock configuration and connection to their reference clocks and networks be eliminated. It would require that all network paths have exactly symmetric delays.

Figure 5.2 shows sample output from a program implementing the computation of predicted time. Note that all of the updates to the stored systematic offsets are small relative to the values of offsets involved.

### 5.2.5 Determining when integrated `predrsadj` is meaningful

Sometimes there is no reason to believe that the result of the computation of integrated predicted time is useful. The computation checks for these circumstances and includes in its output a valid bit. If the output is not valid, rather than setting `rsadj` equal to `predrsadj` for the next time period, an alternative action is taken. Possible choices include letting the system clock be controlled by the standard NTP algorithm, or by the previous valid computation of integrated predicted time.

The computation of integrated predicted time is considered invalid if

1. the set of observations to be considered is empty, or

2. the set of observations to be considered spans less than a configurable amount of time (8 hours), or

3. the last run covered by the observations is not the current run.

The first condition is obvious, and the second straightforward — integrated predicted time computed over a short period of time cannot be trusted. The third condition is also straightforward — predicted time computed for a given run is of little use for setting `rsadj` for the next run, since the phase is unknown.

## 5.3 Implementation

As explained earlier, the program computing predicted time takes as input values of `uoffset` and `rsadj`, contained in log messages generated by the NTP daemon. The integrated timekeeping scheme is implemented by the same program running in "prediction" mode rather than "analysis" mode. Periodically new log messages are obtained and read, and a value of `predrsadj` is output. This value is conveyed to the running NTP daemon via an NTP control message. Obviously the analysis and prediction program could be integrated into the NTP daemon, avoiding a great deal of this complication and potential source for run-time problems. However, debugging of the prediction program would be much more difficult and time-consuming, since the NTP daemon would be restarted every time a change were made.

I included a simple sanity check in the timekeeping daemon for use when running the integrated scheme. If 30 minutes pass since the last time the NTP system offset took on a value smaller in magnitude than a configurable constant (25 ms in the implementation used in this research), the running of the integrated scheme is temporarily terminated, and the standard NTP algorithms once again control the system clock until another control message is received. In a production system, additional sanity checks would be appropriate. For example, a new value of `predrsadj` that would cause the current system offset to be greater in magnitude than this error constant should be rejected, and the standard NTP algorithms run. In this way, gross errors in the computation of integrated time will be ignored, and the integrated scheme conditioned by sanity checks cannot perform grossly worse than the standard NTP algorithms.

## 5.4 Simulation

I desired to directly compare the behavior of the standard NTP algorithms and the integrated timekeeping scheme. To do this, I implemented a simulation of the prediction scheme. The actual implementation computes integrated predicted time periodically, and then directs the NTP daemon to steer the system clock according to the resulting value of `predrsadj`. The simulation computes integrated predicted time, and computes the value that `rsadj` would have had if a daemon had been commanded to set `rsadj` = `predrsadj`. The new value is called `simrsadj`, for *simulated* running sum of adjustments. Thus, the behavior of the NTP algorithms — which actually were run while the offset observation data were being gathered — may be analyzed. Many versions of the integrated timekeeping scheme can be tried without disturbing either the quality of time on the experimental host, or, more importantly, the data collection process for these experiments.

The program implementing the simulation first reads data covering the time period of the simulation. Then, for times beginning near the start of data and becoming progressively greater, the program computes the value of `predrsadj` which would have been computed at that time. Then, rather than instructing the running daemon to set `rsadj` equal to this value of `predrsadj`, `simrsadj` is set to `predrsadj` for times between the current simulation time and the next time integrated predicted time is to be computed. After this has been done for each time at which the computation of predicted time would have occurred, the analysis algorithm is run, and the resulting value of `predrsadj` used to analyze the behavior of the local oscillator and remote clocks. It is also used to analyze the behavior of the synchronization algorithm that was actually run (generally NTP) while the data were taken.

Because the `rsadj` measurement technique only relies upon values of `uoffset` and `rsadj`, the analysis method can be correctly applied to behavior that did not actually happen, but was simulated. The value of `predrsadj` computed for analysis at the end of the simulation is also used to analyze the behavior of the integrated timekeeping scheme by examining the value of `simrsadj` with respect to the final value of `predrsadj`:

$$\texttt{simrsadjresid} = \texttt{simrsadj} - \texttt{predrsadj}$$

With the simulation, it is possible to directly compare the behavior of the standard NTP algorithms with the integrated timekeeping scheme. Since the NTP algorithm actually did run on the experimental computer, `rsadj` is $\texttt{rsadj}_{\texttt{NTP}}$, and `simrsadj` is the value `rsadj` would have had if the integrated timekeeping scheme had actually been run. It also becomes easy to make comparisons of different versions of the integrated timekeeping scheme, e.g., the standard version and one which does not discard high-delay samples. The above comparisons are meaningful because they directly compare the behavior of each scheme when presented with identical network offset measurements and local oscillator behavior.

## 5.5 Smoothing of `rsadj` during integrated timekeeping

Graphs of `rsadjresid` for the integrated timekeeping scheme actually being run and being simulated will be shown. While generally similar, there is one significant difference. When the scheme is simulated, discontinuities are present at the times the estimation is performed. This is to be expected; rsadj is at those times set to a new value of predrsadj, and the estimated phase changes slightly because of the new data.

When the integrated timekeeping scheme is actually run, however, some smoothing of rsadj occurs. When the daemon is commanded to set rsadj equal to predrsadj, it instead periodically computes predrsadj, compares it to rsadj, and adjusts the clock by a fraction of the difference. In this way, rsadj approaches the correct value, and does not exhibit discontinuities — although its derivative does.

Another approach would be to set rsadj to be the result of linear interpolation between the current value of rsadj and the value of predrsadj as of the next scheduled running of the computation of predicted time. Or, perhaps a higher-order polynomial should be used for interpolation, so that not only phase but frequency is continuous.

I contend that the appropriate behavior is application-specific, and the user of time should be able to choose which of the above behaviors is desired. In the absence of a specific choice on the part of the user, the exponential decay to the specified value of predrsadj is easy to implement and appears reasonable.

## 5.6 Analyzing integrated timekeeping with the rsadj technique

Just as I analyzed the synchronization behavior resulting from the standard NTP algorithms controlling the system clock with the rsadj measurement scheme, I can examine the synchronization behavior resulting from running — or simulating — the integrated timekeeping scheme.

The top graph of figure 5.3 shows values of rsadj for garlic for a period when the system clock of garlic was being controlled by the standard NTP algorithms. The middle graph shows values of simrsadj, or simulated rsadj, from a simulation of the integrated timekeeping scheme. The bottom graph shows a close-up view of simrsadj, indicating that the view of the system clock with respect to the local oscillator was stable.

## 5.7 Analyzing integrated timekeeping with a trusted reference

I can also use the trusted reference method to verify simulated behavior. While the value of capture describes the behavior of the synchronization algorithms actually run rather than simulated, I can compute

$$\texttt{simcapture} = \texttt{capture} - \texttt{rsadj} + \texttt{simrsadj}$$

This is the value of capture that would have been observed had the computer actually been running the integrated timekeeping scheme.

Figure 5.4 shows values of capture and simcapture over the same time period as Figure 5.3. In comparing values of simcapture and simrsadj, I find that the small phase discontinuities every time a new value of predicted time is computed are only visible in the graph of simrsadj, and are obscured by the ethernet noise in simcapture. The graph of simcapture, however, shows some wander over periods of several days, such as from March 8 to March 15. Thus, it is helpful to examine both simrsadj and simcapture in order to measure the performance of the integrated scheme.

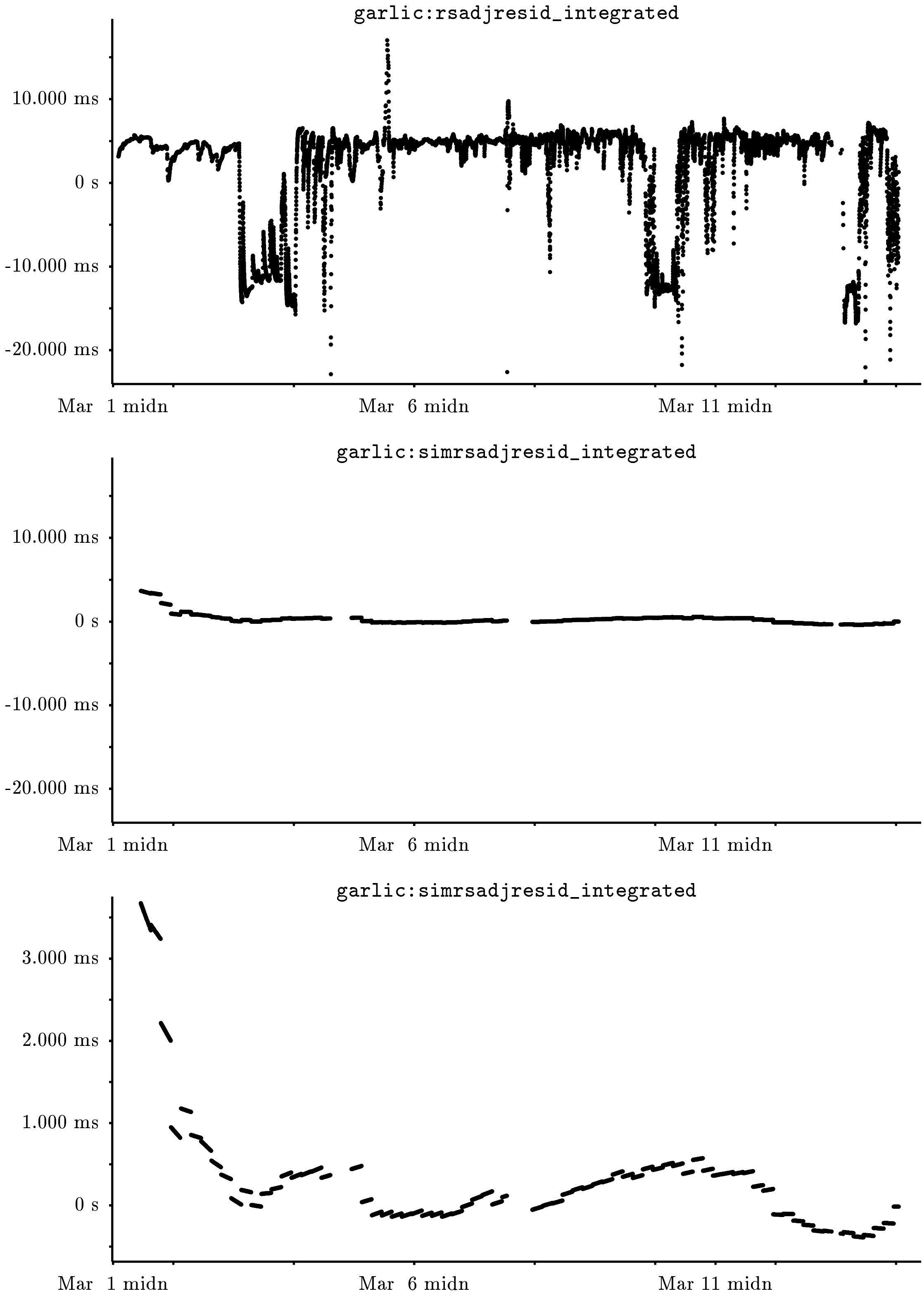



Figure 5.3: `rsadj` for garlic running NTP, above, and simulated `rsadj` for the integrated timekeeping scheme, middle and below.

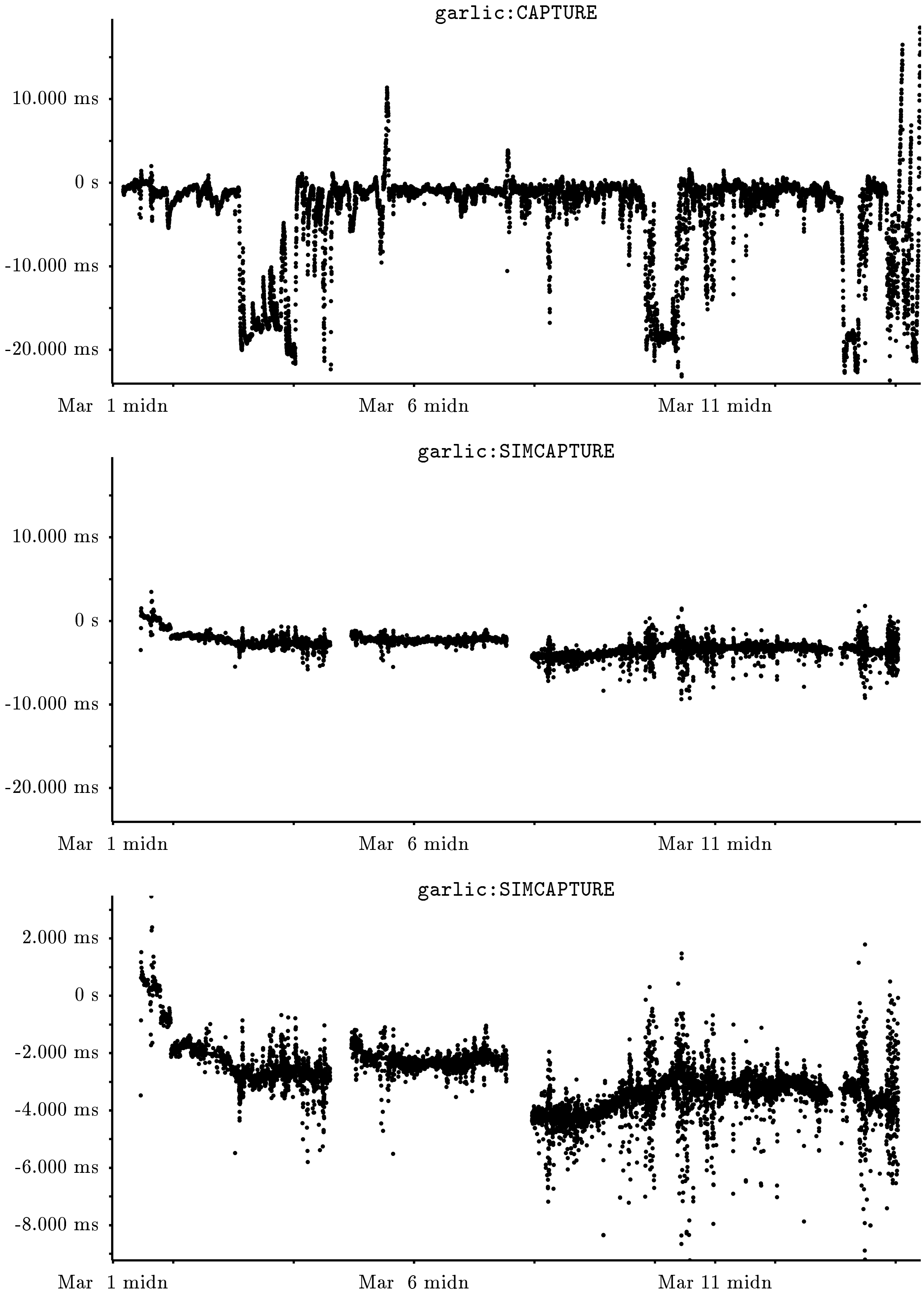



Figure 5.4: `capture` for garlic running NTP, above, and simulated `capture` for the integrated timekeeping scheme, middle and below.

## 5.8 Performance analysis of integrated timekeeping

I compare the performance of the integrated timekeeping scheme to that of the standard NTP algorithms by actually running the NTP algorithms and simulating the integrated scheme on the same data. I have found that the choice of interval for the integrated timekeeping scheme affects the performance, and that choosing an interval that is small enough is critical. I examine the performance of the scheme with some features disabled, in order to understand their impact on performance.

Figures 5.3 and 5.4 show the behavior of the integrated timekeeping scheme under typical conditions of network and remote clock behavior. Examining the graph of `simrsadjresid`, the view of the simulated values of the system clock relative to the local oscillator, one can see that once the integrated scheme had available at least several days of data, values of time were stable *with respect to the local oscillator* to within roughly a millisecond. However, this implicitly assumes that the local oscillator was perfectly stable, and this assumption is the foundation of the integrated scheme. Thus, it should not be surprising that the such a result is obtained, and one cannot rely on such a result to claim that the integrated timekeeping scheme works well. Nevertheless, the observed reasonable long-term stability argues that the local oscillator was very stable over much shorter periods, and one may reasonably interpret this data as indicating that over short time periods the value of the system clock was stable.

Examining the graph of `simcapture`, the view of the simulated system clock relative to GPS-derived time, one can see that in fact the value of the system clock was stable to within 2 or 3 ms. Comparing this graph to that of `simrsadjresid`, several features are strikingly different. The small changes in `simrsadjresid` occurring when new updates are computed are obscured by the ethernet noise in the graph of `simcapture` resulting from transferring GPS time from ipa to garlic. However, the trend line showing the value of the system clock can be shown clearly in the graph of `simcapture`; one merely has to assume that the local oscillator was stable over periods of an hour or so. This trend, however, differs from the values of `simrsadjresid` in two important ways. One is that there appears to have been an unmodeled change in the local oscillator near midnight on March 12. Another is that there appears to be a discontinuity on March 7; this is due to a reboot of garlic. During the next few days, accurate information about local oscillator frequency is available from before the reboot, but less accurate information is available about phase, since garlic does not have a mechanism to preserve phase across reboots.

Figure 5.5 shows six regions. In the first, the standard NTP algorithms were run. In the second, the integrated timekeeping scheme was run, using most of the remote clocks used by the NTP algorithms that exhibited reasonable behavior. In the third region, the integrated timekeeping scheme was run, but the remote clocks used by the scheme were restricted to one clock in Norway, one in Switzerland, and one in Australia, chosen for their high-delay and high-variance paths. In the fourth, a single estimate was used for several days. In the fifth, another estimate was used, but entered intentionally with a 20 ms phase error. In the sixth, the standard NTP algorithms were again run, but with only the three clocks used in the third interval.

By comparing the third and sixth intervals, I can compare the performance of the standard NTP algorithms and the integrated scheme on similar but not identical data. It

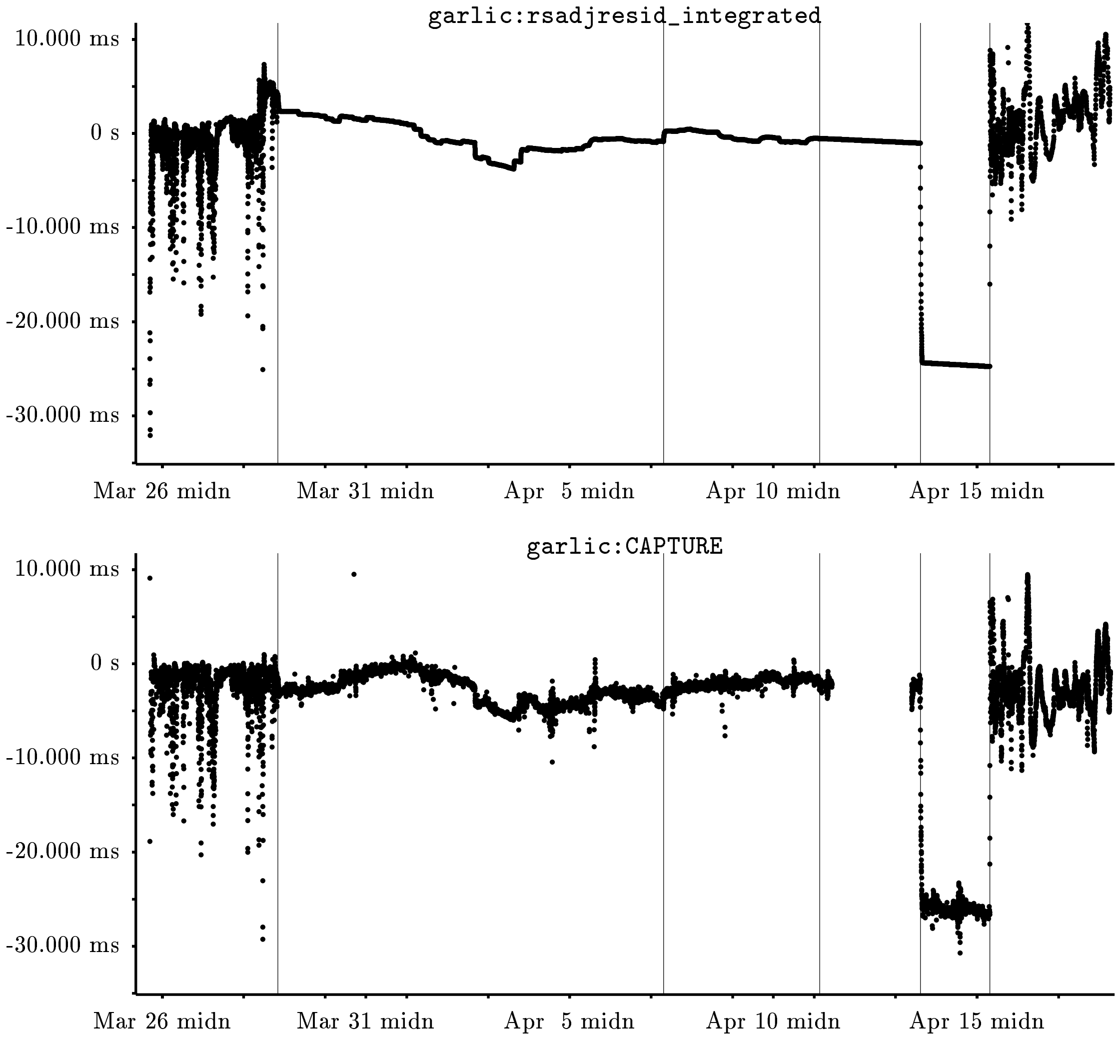
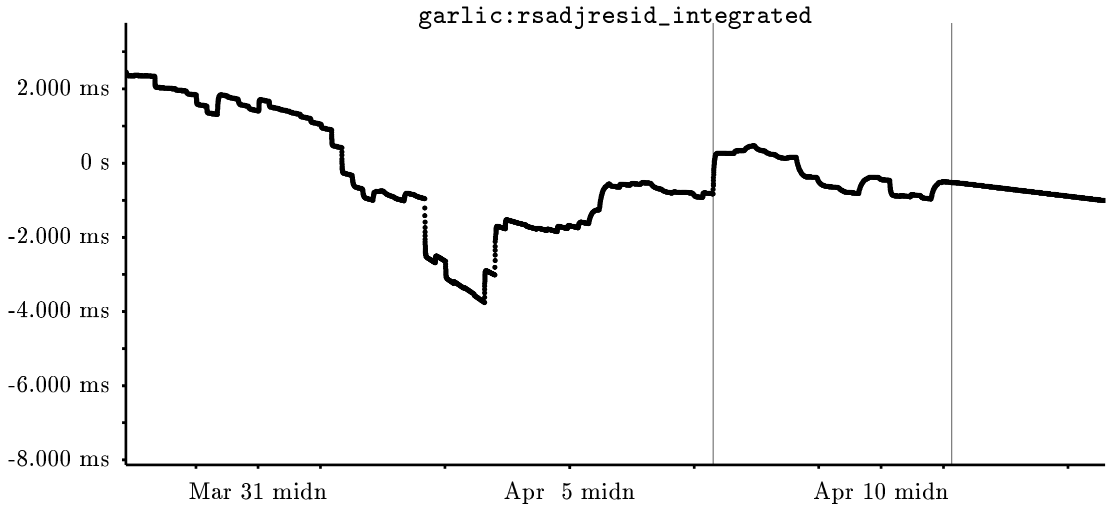



Figure 5.5: Above, values of `rsadjresid` vs. time for `garlic` running the standard NTP algorithms, integrated timekeeping with a set of "good" remote clocks/network connections, integrated timekeeping with clocks in Europe and Australia, following two estimates for several days each, and again with NTP. Below, the corresponding values of `capture`. The regions are separated by vertical lines.

can be clearly seen that local timekeeping accuracy is strikingly better with the integrated scheme.

It is also interesting to compare the first and sixth intervals. NTP appears to have performed better with the distant clocks than with the normal complement. I believe that this is because while the European and Australian clocks exhibit greater network delay variance, they have lower systematic offsets among them. Note also that while the fifth interval, showing the results of following a single value of predicted time that is in error by 20 ms, is clearly different from the behavior of NTP in the first interval, there were some values of both `rsadj` and `capture` recorded in the first interval that differed from the correct values to the same extent. In contrast, no such values were recorded in the intervals



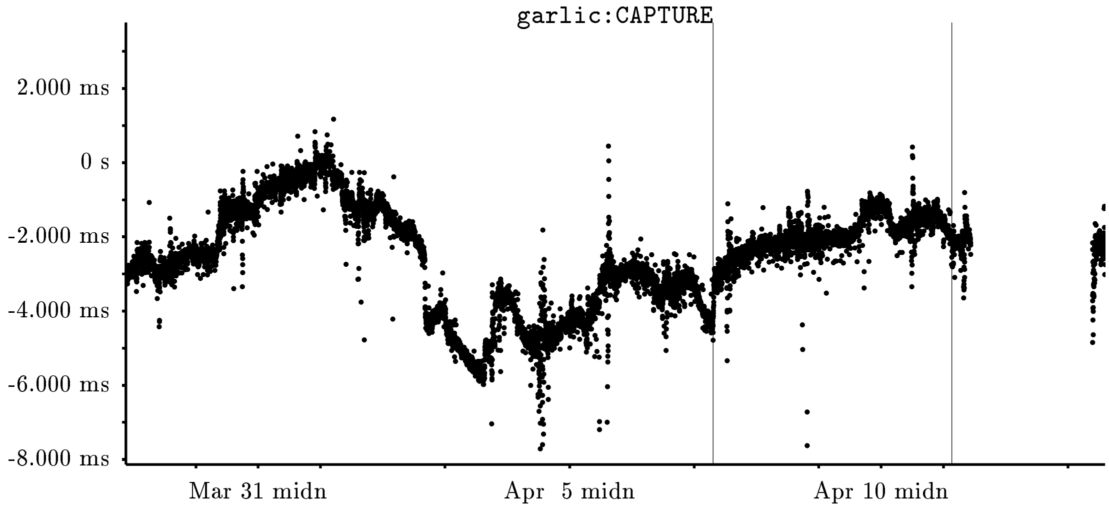



Figure 5.6: Above, values of `rsadjresid` vs. time for garlic running integrated timekeeping with a "good" set of clocks, with clocks in Europe and Australia, and following an estimate for several days. Below, the corresponding values of `capture`.

in which integrated timekeeping was being run.

Figure 5.6 shows a close-up view of the second, third and fourth intervals shown in the previous figure. An interesting feature is evident in the graphs near midnight April 3; the system clock underwent a phase excursion of roughly $-3$ ms for approximately a day. I believe that this was due to a change in the time period over which data from one or more of the remote clocks was used, causing a change in systematic offset. Examining these graphs, one can also see that the graphs of `rsadj` and `capture` do not correspond exactly, due to ethernet delay variance and unmodeled frequency errors in the local oscillator.

Figure 5.7 shows a closeup of garlic being synchronized with the integrated scheme to the three distant remote clocks, following a single estimate, following the incorrect estimate (not visible due to scale), and being synchronized by NTP to the same three remote clocks.

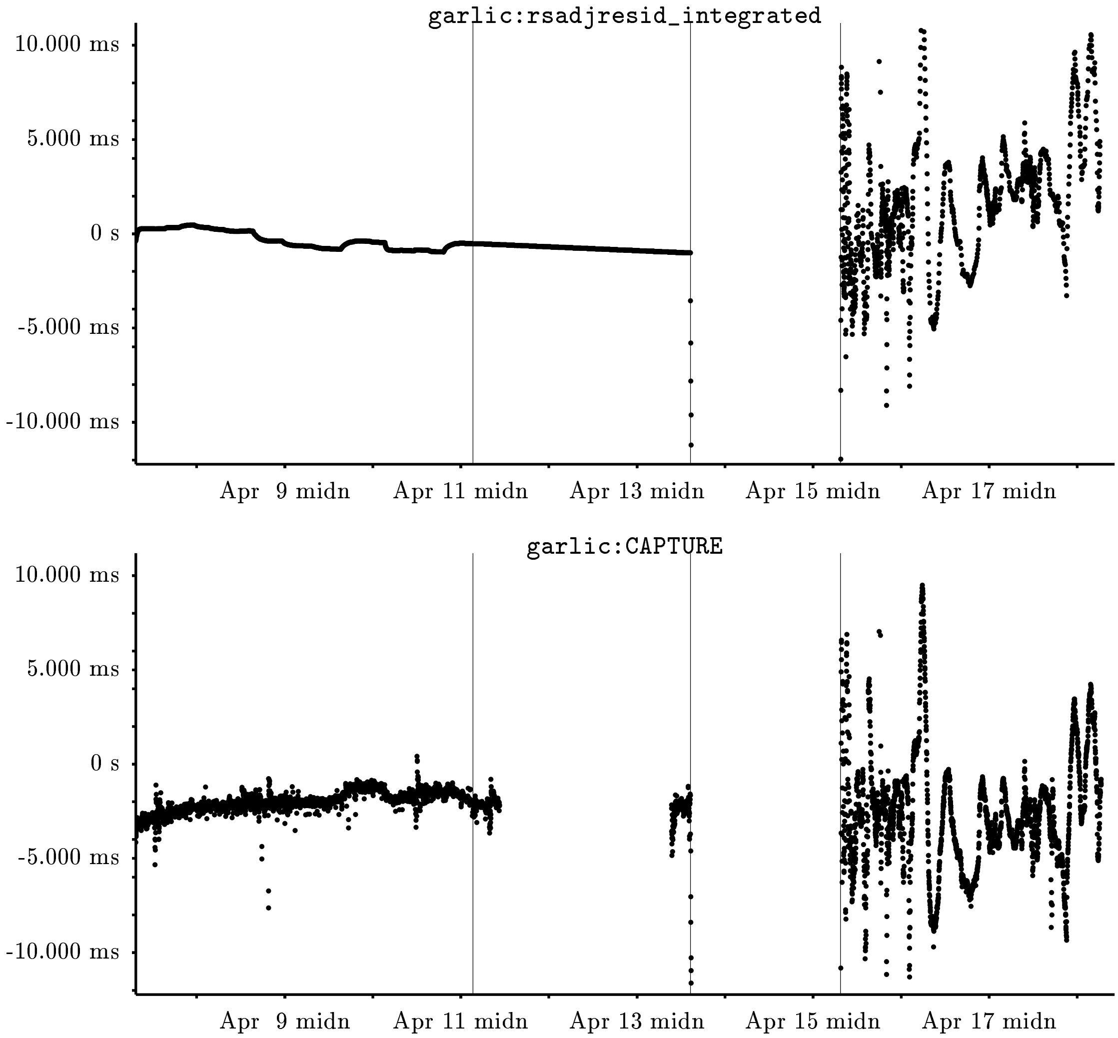



Figure 5.7: Above, values of `rsadjresid` vs. time for garlic integrated timekeeping with clocks in Europe and Australia, following an estimate for several days, and following the clocks in Europe and Australia with NTP. Below, the corresponding values of `capture`.

Examination of the graph of `capture` shows that while being synchronized by NTP, the average offset relative to GPS time was small compared to the variance. With the integrated technique, the average offset relative to GPS is similar, but the variance is much smaller.

Figure 5.8 shows the behavior of garlic under the NTP algorithms, top, and the integrated scheme, middle and bottom. The local oscillator at the time was a Bliley OCXO. The integrated scheme seems to perform poorly at the extreme left portion of the graphs. This is in fact the case, and is typical of running the integrated scheme without prior data. Once there was enough data from which to reliably estimate clock parameters, the performance improved significantly. This startup transient behavior is analogous to starting NTP without the benefit of a stored frequency estimate, such as the first time it is run on a computer. While interesting, such behavior is not typical of the use of an algorithm for

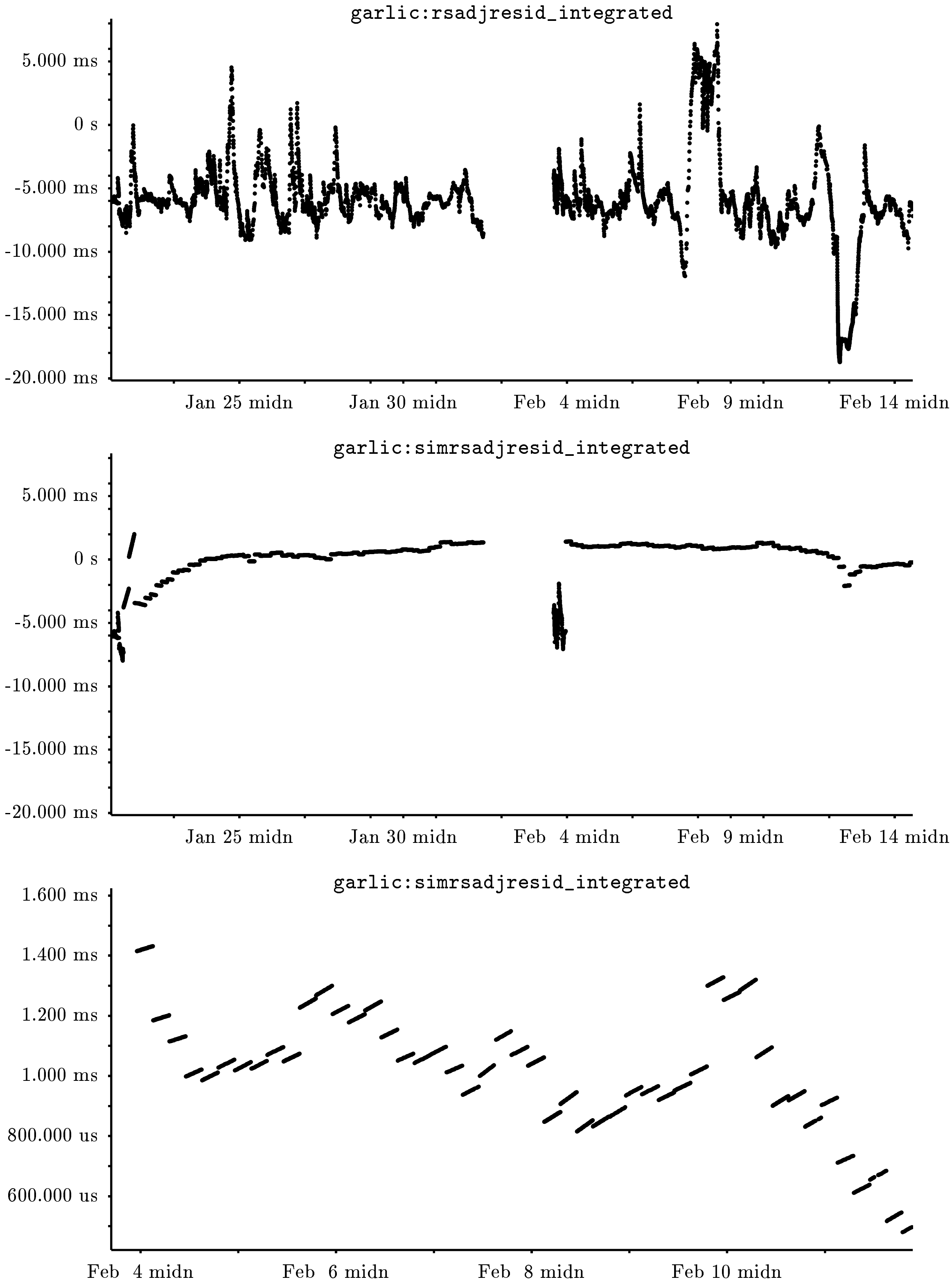



Figure 5.8: Performance of NTP, above, and the integrated timekeeping scheme, below, on garlic viewed with the `rsadj` technique.

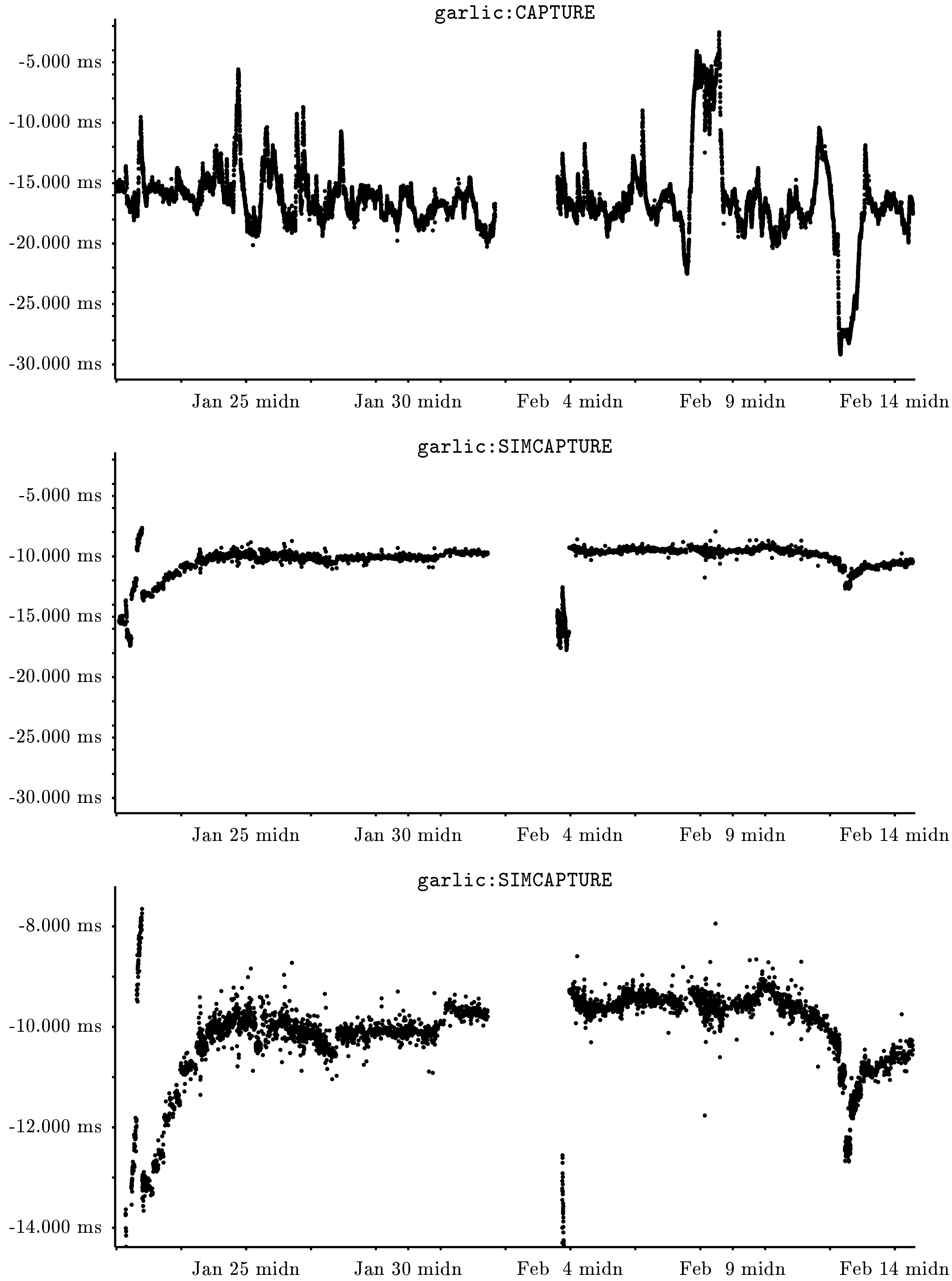



Figure 5.9: Performance of NTP and the integrated timekeeping scheme on garlic, viewed with the trusted reference technique.

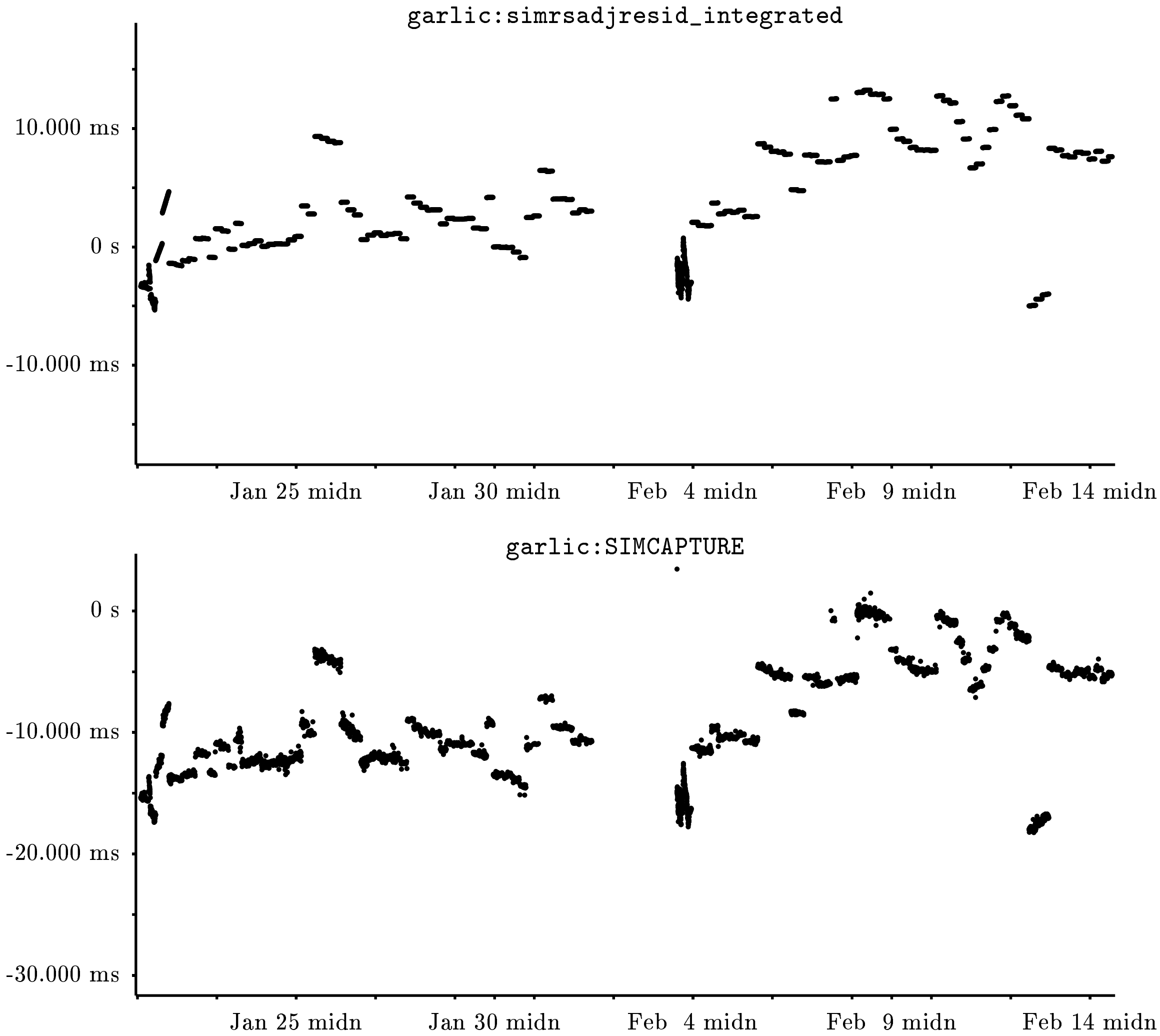
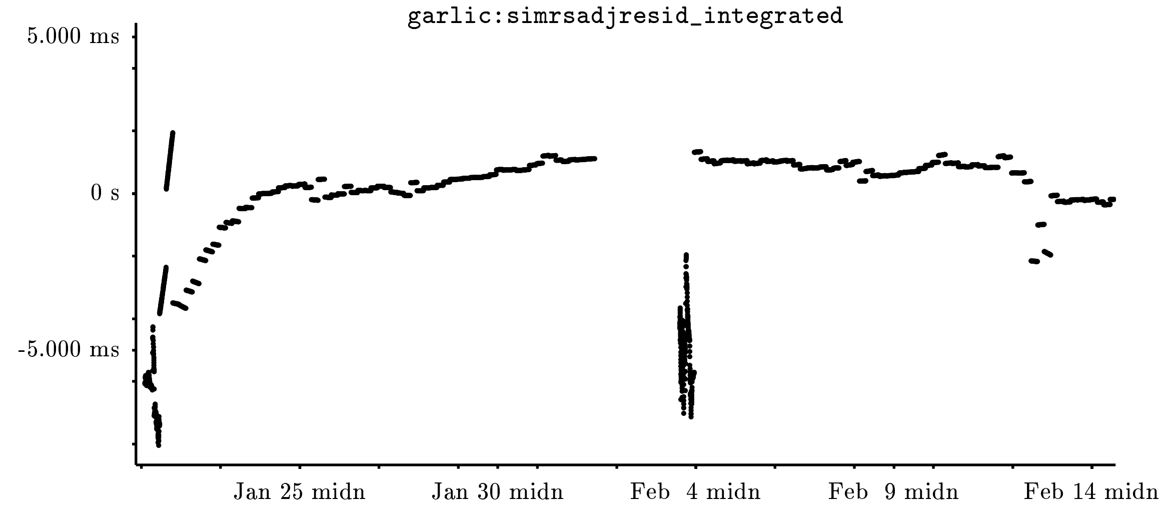



Figure 5.10: Performance of integrated timekeeping without systematic offset estimation on garlic viewed with the `rsadj` and trusted reference techniques.

synchronization in the long term.

The center of the graph shows a short period on February 3 where the integrated scheme declined to control the system clock. This time period is from when garlic was turned on after being off for several days. Just after February 12 at midnight, NTP caused the system clock to undergo a negative phase excursion of roughly 13 ms. The corresponding time in the simulation shows only a roughly 2 ms excursion. The bottom graph shows a closeup of `simrsadjresid` for roughly a week near the middle of the upper graphs. The phase discontinuities between successive values of `predrsadj` generated by the integrated scheme are very small, and most are on the order of 100 $\mu$s. Figure 5.9 shows `capture` rather than `rsadjresid` for the same time period as Figure 5.8.

Figure 5.10 shows the performance of the integrated timekeeping scheme without the step to estimate and remove systematic offsets. The same time period as in Figure 5.8 is



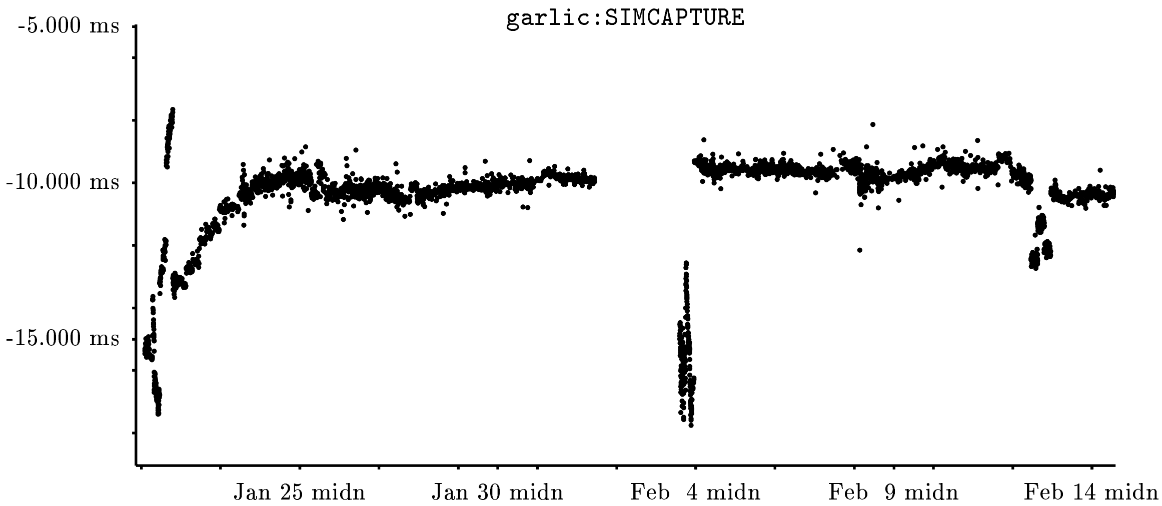
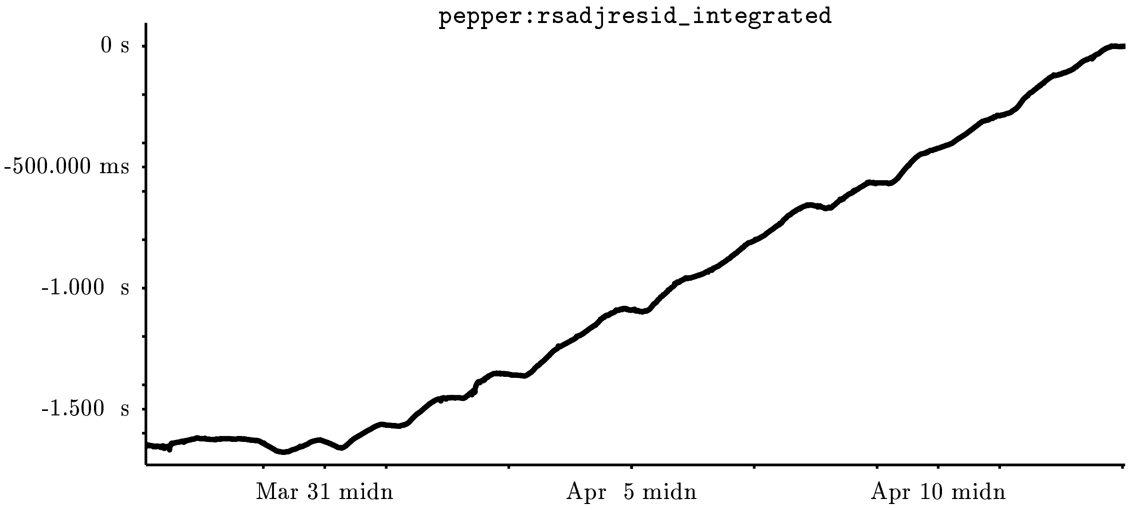




Figure 5.11: Performance of integrated timekeeping without hysteresis in the last interval of the piecewise computation on garlic viewed with the `rsadj` and trusted reference techniques.

shown. Over many short periods of time only small phase variations occur, due to data from remote clocks being used the same proportion in the estimation. However, some phase discontinuities result from changes in the weights of the various remote clocks and in the relative sizes of the last intervals computed for each remote clock.

Figure 5.11 shows the performance of the integrated timekeeping scheme with the constraints on changes of the last interval produced by the piecewise computation of predicted time removed. More and larger phase discontinuities occur than with the full integrated scheme, but performance is better than the integrated scheme without systematic offset estimation.

Figure 5.12 shows the results of pepper running the standard NTP algorithms and the integrated scheme with a 90-minute update interval. pepper is a VAXstation 3200 with the supplied local oscillator. While the integrated scheme produced roughly the same behavior



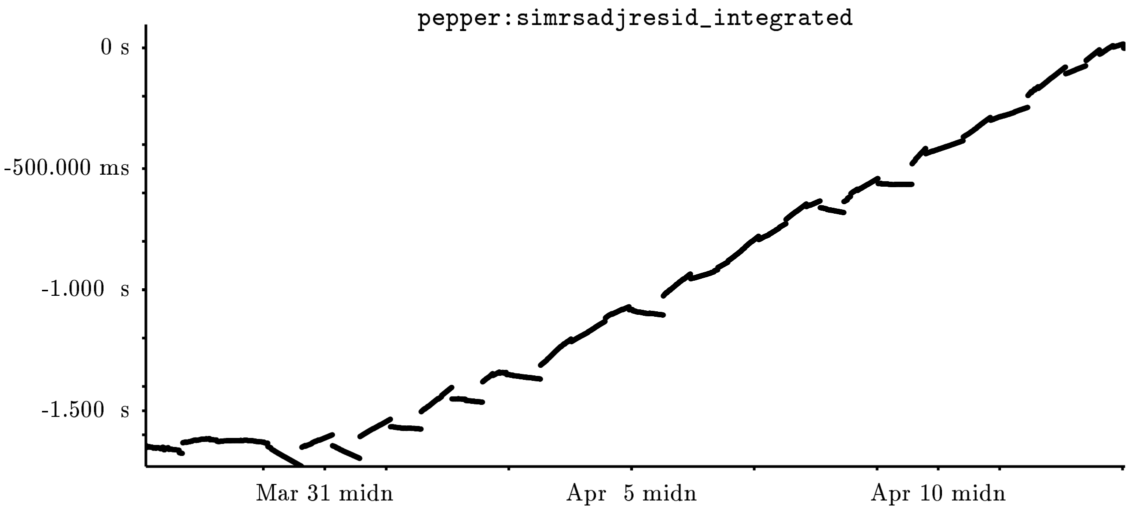




Figure 5.12: Above, values of `rsadjresid` for `pepper` running the standard NTP algorithms. Below, simulated values running the integrated timekeeping scheme with a 90-minute update interval.

as NTP, it is difficult to compare the graphs because the poor performance of the local oscillator causes a large vertical scale to be required.

Figure 5.13 shows values of `aoffset` for `pepper` being controlled by NTP and corresponding values of `simaoffset`, the actual offsets that would have resulted had the integrated timekeeping scheme been run. Because the vertical scale is smaller, a meaningful comparison can be made. Because `noc.near.net` is normally synchronized to far better than the errors shown in the graph of `simaoffset`, examining values of `simaoffset` is reasonable. NTP provides far superior timekeeping; the changes in frequency of the local oscillator are too frequent and significant to allow extrapolation of predicted time. The bottom graph shows values of `simaoffset` when the integrated scheme is run with a 20-minute update interval. While the magnitude of errors in time is slightly lower, the performance is still far inferior to that of NTP. In this case, the assumption that the recent behavior of the local

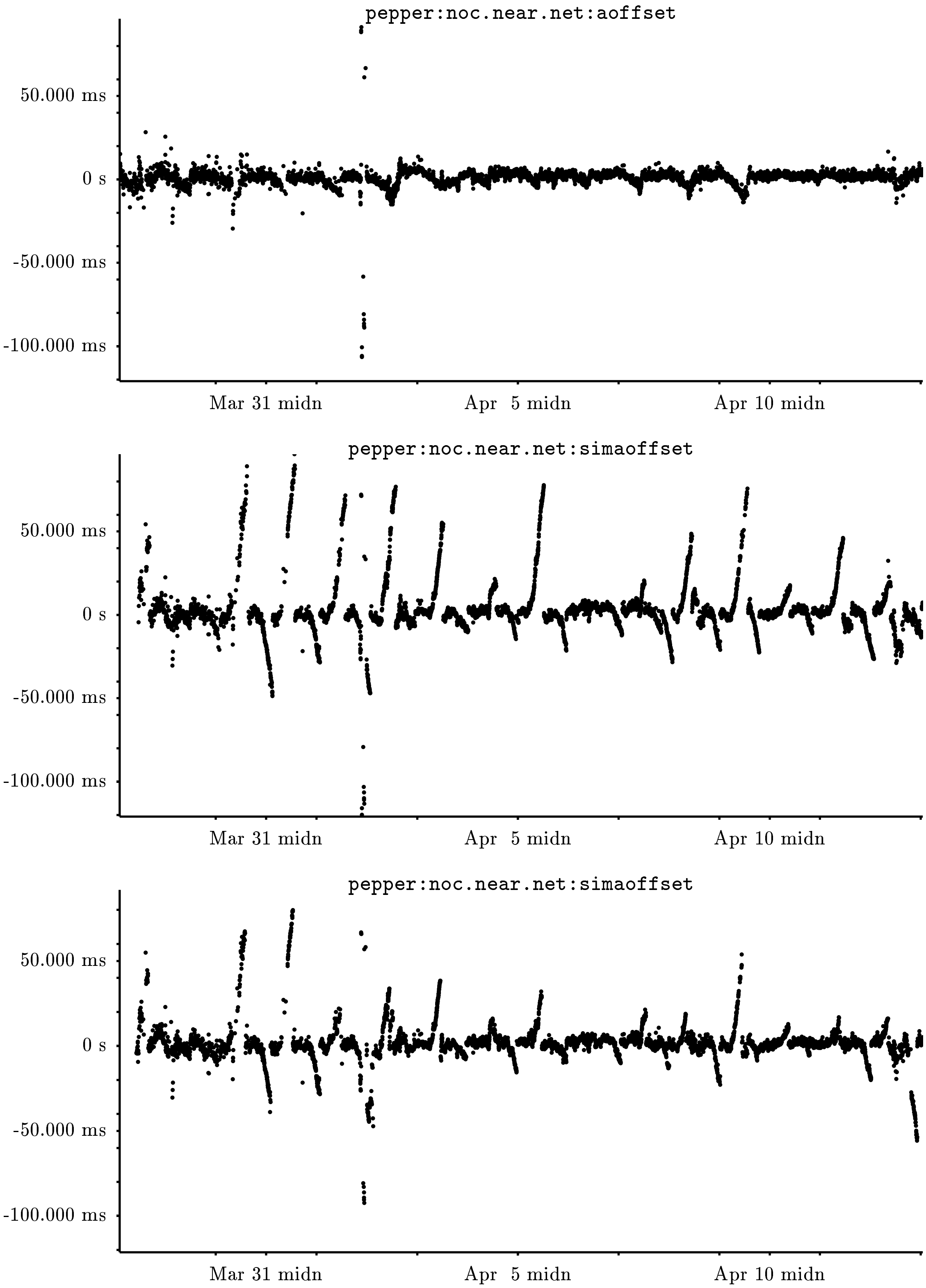



Figure 5.13: Above, values of `aoffset` of noc.near.net as observed by pepper running the standard NTP algorithms. Middle, simulated values running the integrated timekeeping scheme with a 90-minute update interval. Below, simulated values with a 20-minute update interval.

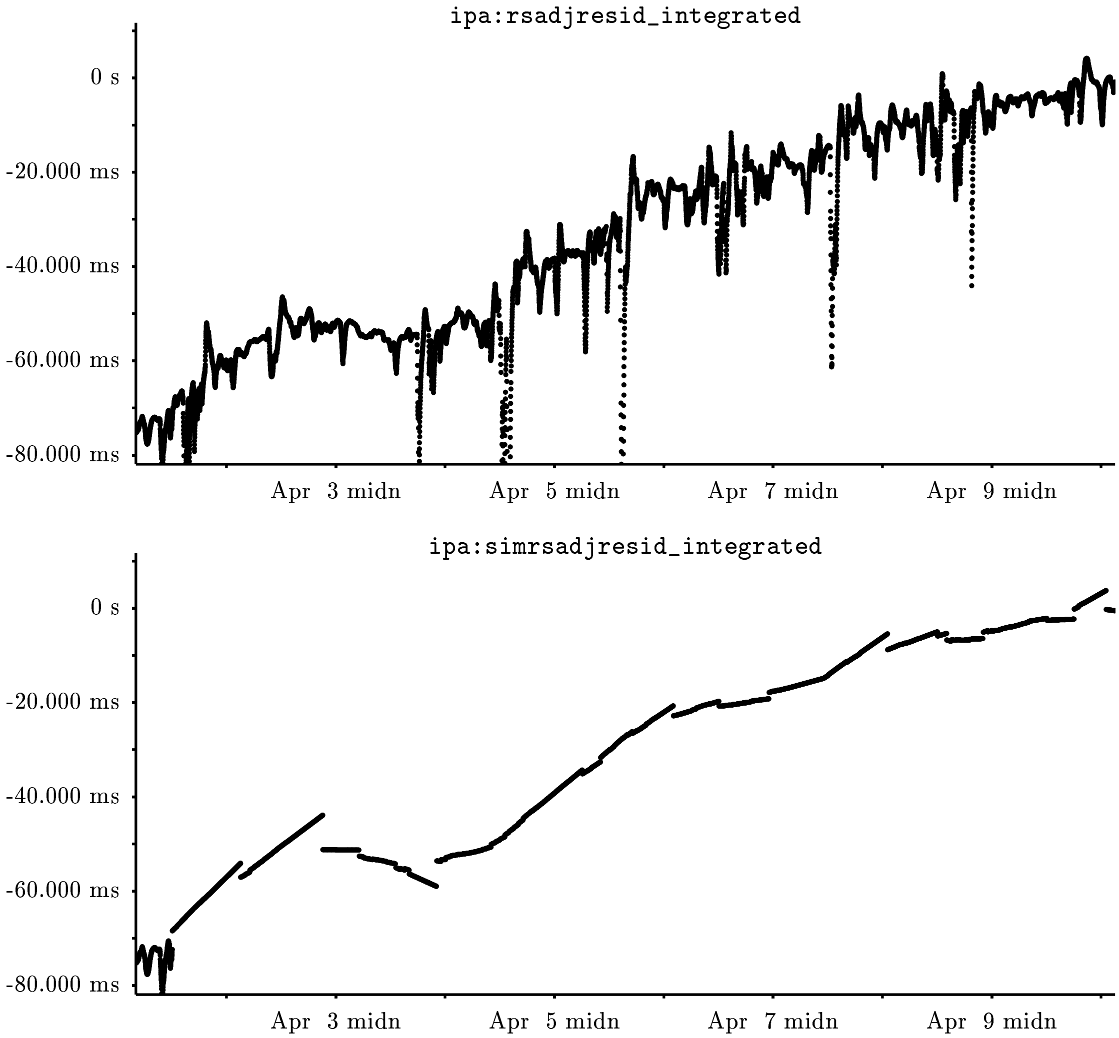



Figure 5.14: Performance of NTP and the integrated scheme on ipa.

oscillator is a good predictor of its behavior in the immediate future is not true.

Figure 5.14 shows values of `rsadjresid` for ipa being synchronized by NTP, and simulated values for ipa being synchronized by the integrated scheme with an update interval of 1 hour. As with the graphs of `rsadjresid` and `simrsadjresid` for pepper, the large vertical scale makes comparison difficult.

Figure 5.15 shows `capture` data for ipa. Here, it can be seen that the integrated scheme in fact performed better than NTP. However, the bottom graph shows that the integrated scheme used values of `predrsadj` beyond the time that such values actually characterized the clock. On April 2, the simulated system clock became in error by roughly 5 ms relative to the average value.

This comparison, is, however, highly misleading. ipa was at the time configured to make offset observations of garlic and larch, and NTP used both remote clocks. The system clock of larch does not keep time well, occasionally exhibiting large phase discontinuities. Thus,

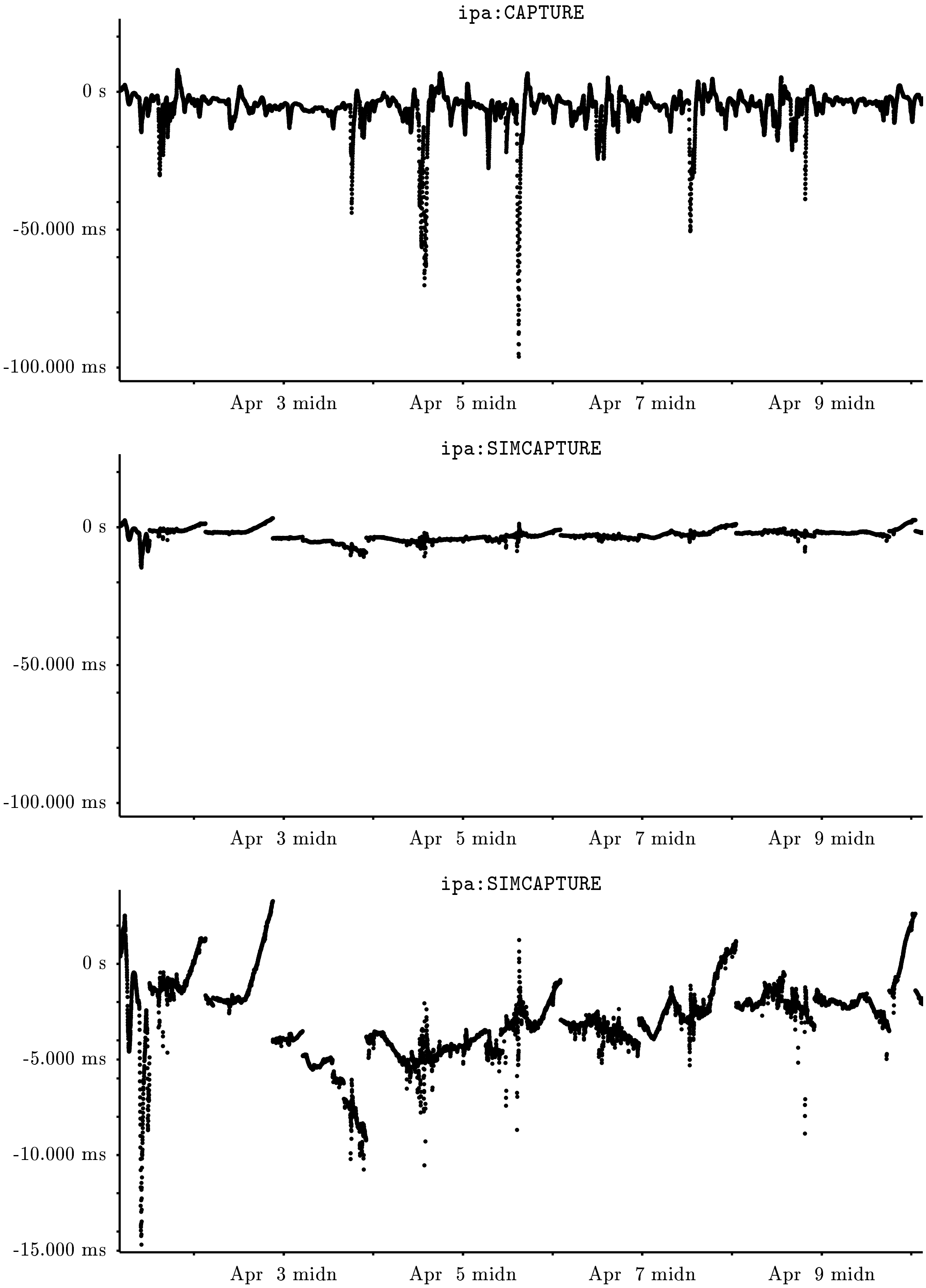



Figure 5.15: Performance of NTP and the integrated scheme on ipa, viewed with the trusted reference technique.

NTP running on ipa exhibits large amounts of short-term phase variation as it switches between garlic and larch.

The integrated scheme, however, was configured to use only offset observations from garlic. Thus, the results are not directly comparable. They do, however, serve to illustrate that with a good-quality remote clock and the local oscillator used by ipa — an OCXO which has apparently failed and now exhibits large frequency variations — the integrated timekeeping scheme performs reasonably.

## 5.9 Stability of the integrated timekeeping scheme

One claim made by the Network Time Protocol specification is that the protocol is stable when multiple computers running the protocol each derive time from the next. Thus the question naturally arises as to the behavior of the integrated timekeeping scheme when tracking a remote clock that is being synchronized by the integrated scheme, rather than by NTP. Also, one might ask how well the integrated scheme performs when observing a remote clock that is synchronized by NTP watching other clocks over the network, rather than to an attached reference clock.

Figure 5.16 shows a comparison between timekeeping obtained by the standard NTP algorithms running with multiple remote clocks and timekeeping obtained by the integrated timekeeping scheme observing only one remote clock, psc.edu. The bottom graph shows predicted offsets to psc.edu over the same time period, based on a single value of predicted time. The integrated scheme behaved roughly as well as NTP under these circumstances. Near midnight February 9, the effects of the large negative phase excursion of psc.edu can be seen; the integrated scheme tracked this excursion.

The question arises of the behavior of the integrated scheme if arbitrary numbers of cascaded computers ran it, each synchronizing to the next. This is unfortunately a very difficult question to answer. It is my belief that the scheme is stable as long as each computer using the scheme produces reasonable results — that is, has a value of `rsadj` such that `rsadjresid` has reasonably small values and is reasonably smooth. If, on the other hand, the quality of the local oscillator is poor, the value of the system clock will often be incorrect, as the system will set `rsadj` to a value of `predrsadj` which no longer describes the behavior of the local oscillator. In such cases, the degenerative behavior of the next computer estimating incorrect parameters of its local oscillator based on the incorrect time of the remote clocks it is observing seems likely.

Figure 5.17 shows the results of ipa running the NTP algorithms and making observations to `garlic` and `larch`. To the left of the vertical bar, garlic's clock was controlled by NTP, and to the right it was controlled by the integrated scheme. Figure 5.18 shows the results of simulating the integrated scheme on ipa, using garlic as the only remote clock. Reasonable performance was achieved in both cases.

It might be appropriate to respond to requests for the time via NTP packets on the network with the value of time the standard NTP algorithms would have returned, but to cause the value of the system clock to behave as if the integrated scheme is being run.[1] With this hybrid scheme, the stability properties of NTP are guaranteed, while giving local

[1] Thanks to Tim Shepard for this observation.

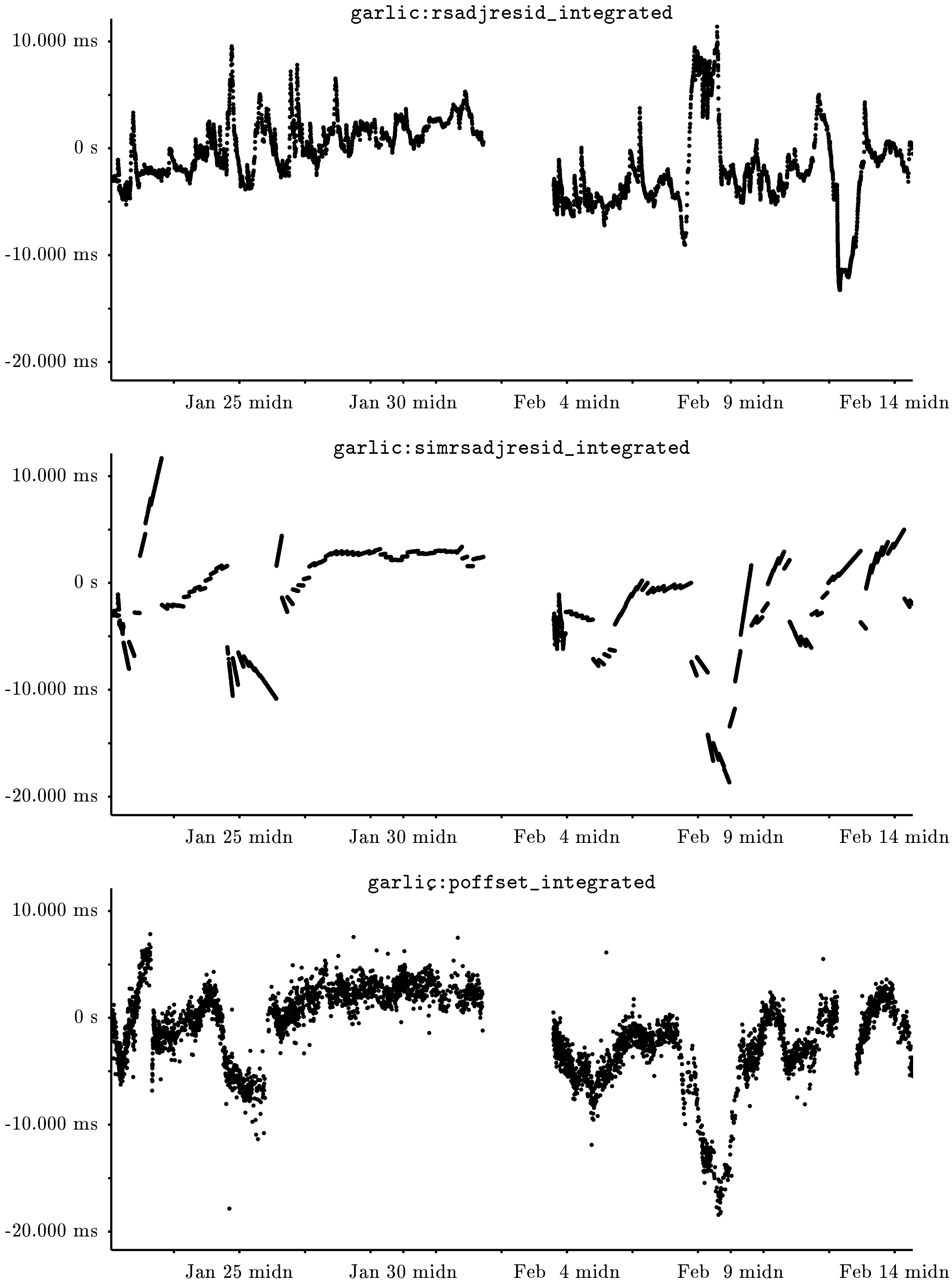



Figure 5.16: Performance of NTP and integrated timekeeping on garlic viewed with the `rsadj` technique when the integrated scheme followed a single remote clock, top and middle. Predicted offsets to the remote clock, bottom.

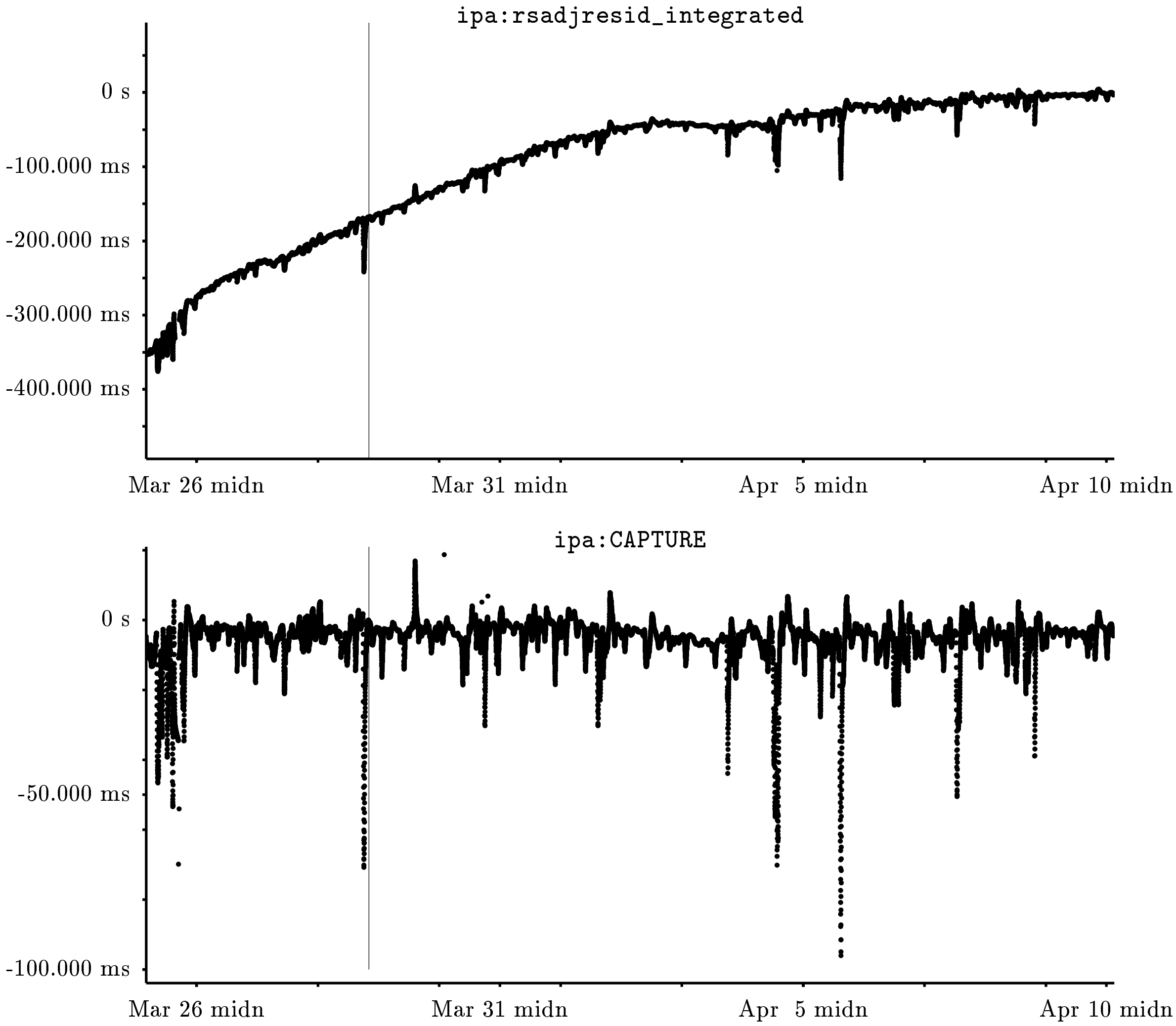



Figure 5.17: Performance of NTP on ipa tracking garlic and larch, while garlic ran the NTP algorithms, left, and the integrated timekeeping scheme, right.

time the benefit of the integrated scheme. However, computers obtaining time would not have the benefit of the more precise timekeeping produced by the integrated scheme.

## 5.10 Summary

The integrated timekeeping scheme is a mechanism to calculate values of the system clock based on values of the local oscillator and estimates of the local oscillator's parameters, and a procedure to estimate periodically the parameters of the local oscillator. The performance of the integrated scheme depends critically on the performance of the local oscillator.

It is important to consider the costs of running the integrated timekeeping scheme. They are clearly far greater than those of NTP, in that perhaps several weeks of past observations

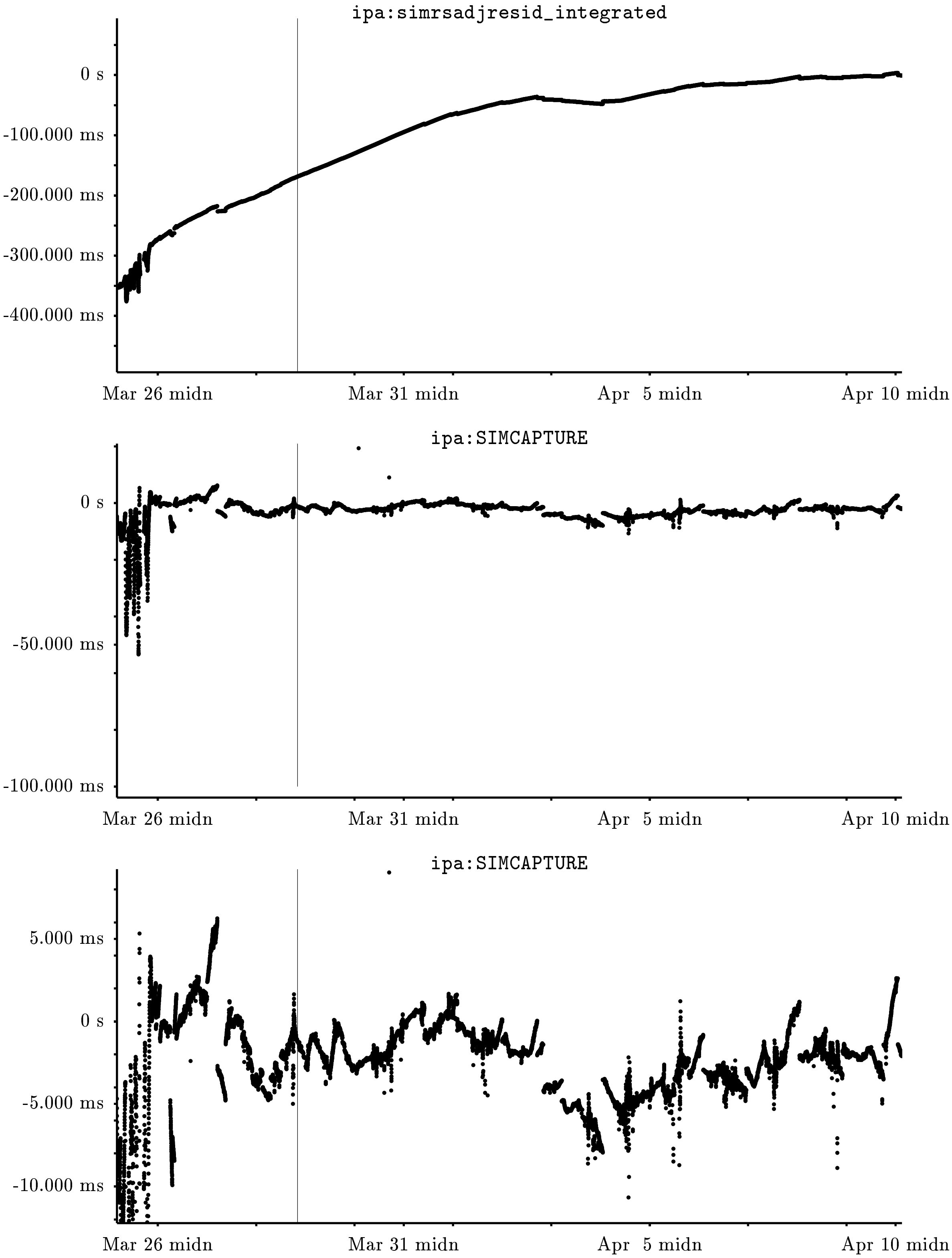



Figure 5.18: Performance of the integrated timekeeping scheme on ipa tracking garlic, while garlic ran the NTP algorithms, left, and the integrated timekeeping scheme, right.

must be retained, rather than merely the last 8 for each remote clock. Periodically, a computation using these past observations must be performed. The implementation of integrated timekeeping constructed for this thesis did not prove burdensome in practice. Several weeks of integrated timekeeping could be simulated in well under an hour on a modern workstation; this is a tiny fraction of the total CPU time in several weeks. Also, it is likely that further efforts to optimize the implementation of the computation would result in significant decreases in the computational effort required. Each day of offset observations requires roughly 10 to 100 kilobytes of storage when ASCII log messages are compressed. I made no attempt to reduce the storage requirements further; far better compression is almost certainly easily achievable. Several megabytes of storage might be required given the current inefficient scheme; this is still a manageable amount of storage for those who care about precise time. I did not attempt to produce a highly efficient implementation, but rather one that was reasonably efficient and flexible enough to be used in conducting experiments.

If the local oscillator changes frequency by several parts in $10^6$ on a daily basis, the integrated scheme performs poorly. If the local oscillator is stable to several parts in $10^8$, so that several days of observations may be used to estimate local oscillator parameters, the integrated scheme performs significantly better than NTP. Under such conditions, values of the system clock are typically stable to most of an order of magnitude better with integrated timekeeping than with the Network Time Protocol. The integrated scheme performs very well even when observing remote clocks over networks with high delay variance.

Given the data presented in this chapter and the previous chapter, I can explain why integrated timekeeping performs better than NTP given a stable local oscillator. One reason is that it is in fact useful to average the effects of network delay variance over longer time periods than NTP does. Another is that it is useful to ignore recent observations if they are believed to be erroneous; NTP provides no such mechanism, and integrated timekeeping does. Integrated timekeeping scheme explicitly accepts that remote clocks will have systematic offsets, and estimates and corrects for those different offsets. Data is combined from multiple remote clocks with no expectation that the systematic offsets are the same. With NTP, data is combined from multiple clocks to form the system offset, and then processed further. Because of this significant difference, integrated timekeeping is adversely affected by differing long-term systematic offsets far less than NTP.

# Chapter 6
# Hardware Clock Requirements

The central model of this thesis is that of a local oscillator which accumulates time, and of a system clock which is set to the correct time. It is not surprising, then, that I believe that the necessary function of timekeeping hardware is to reliably accumulate the output of a stable local oscillator with adequate precision. I point out in this chapter that typical supplied hardware does not meet these requirements.

During the course of the research described in this thesis, I found it necessary to construct a specialized timekeeping module for the DEC DS5000/25 experimental platform, since the timekeeping functionality inherent in that computer is inadequate for the work conducted. I explain in detail why the supplied timekeeping hardware is inadequate, and discuss the design choices made during the design of the specialized module.

I then argue that sufficient hardware to support precise timekeeping is in fact simple, and all components except perhaps a stable local oscillator should be provided by manufacturers of computers. I argue that a module containing a stable local oscillator need not be expensive.

## 6.1 Typical supplied hardware

Virtually all computers today come supplied with basic timekeeping hardware. Most computers contain a battery-backed-up time-of-day clock, often named with the abbreviations TODR (for time of day register) or TOY (for time of year), that keeps time and date with one second resolution, and functions while the computer is not powered. Typically, the system's notion of time is initialized from the TODR at boot time. At shutdown, or when the time is set, the TODR is set from the system clock.

Almost all computers contain a mechanism to generate a periodic "clock" interrupt at some rate such as 60 Hz, 100 Hz, 256 Hz, or 1000 Hz. This interrupt is often used to drive scheduling algorithms; this use is irrelevant to the present work and will not be discussed further. Another use of this interrupt is in timekeeping. In BSD-derived Unix systems, the system clock is implemented by a kernel variable that holds the number of seconds and microseconds since some epoch (usually midnight UTC on January 1, 1970). It should be noted that this number of seconds is as if leap seconds never occurred; leap seconds are addressed in [Mil92b] and are not considered here. On every clock interrupt, the amount of time corresponding to the nominal interval between interrupts — 10,000 $\mu$s for a system with a 100 Hz clock — is added to the system clock. In such a system each clock interrupt is a tick of the local oscillator in the model used in this document.

Some computers provide a way to determine the number of microseconds either since the last interrupt or until the next one. Some computers provide additional counters that simply count microseconds, with very large capacities, so that they only wrap around once per minute or less frequently. On such computers, a tick of the local oscillator is a microsecond, and the system clock is the kernel variable described above. Rather than updating the kernel variable only on clock interrupts, the system also updates it from the more precise counter whenever a system call to obtain the time is made.

### 6.1.1 Problems with supplied hardware design

Typically, the only information relevant to timekeeping that is retained across power outages (either real, or induced by the switch) is the TODR and the time at which the system was shut down, stored on disk. Thus, the time obtained on booting may be in error due to frequency error of the TODR, and will suffer from the limited precision of one second. It is quite clear that better performance than this was not a design goal, and I will not argue that it should be. However, it is a handicap for a user demanding precision timekeeping.

If the computer is of the design where the only timekeeping information provided is the periodic clock interrupt, there are two significant problems in timekeeping. The resolution of timekeeping is limited by the interrupt period. In addition, there is the potential problem of missed interrupts resulting in incorrect timekeeping. While one could argue that a correctly designed operating system will not lose clock interrupts, I have seen many different types of computers do so, particularly under fault conditions, such as memory parity errors or multiple bad blocks on the disk.

### 6.1.2 Problems with supplied oscillator

There is also the problem of the inherent accuracy and stability of the oscillator used to drive the timing. Workstations are typically supplied with inexpensive quartz crystal oscillators. These oscillators are often not sufficient for precise timekeeping. Again, a more suitable oscillator would result in increased cost, and support of precise timekeeping does not appear to be a design goal of typical computer systems.

### 6.1.3 Measurement of external events with supplied hardware

Sometimes one wishes to accurately measure the timing of external events. For example, one might wish to record times of occurrence of events in a laboratory experiment. Also, one might wish to synchronize the computer's clock to an external reference, such as an atomic clock, or a timing receiver (receiving, for example, WWV, CHU, WWVB, GOES, or GPS signals). One technique currently in use is to cause a character to be sent to a serial port by an external signal, and to have the computer take a timestamp when the receive interrupt occurs ([Mil93], pp. 10-15). While this scheme works reasonably well, it is still affected by interrupt latency and variation in interrupt latency.

## 6.2 Clocks used in experiments for this work

Most of the development of the techniques described earlier as well as many of the experiments were done using a DEC VAXstation 3200. In addition to the 100 Hz interrupt, this computer has an additional microsecond-resolution counter. Tim Shepard, a fellow graduate student, and I augmented the system's kernel to utilize the additional counter. On each clock interrupt, the counter is read, and the difference between the current reading and the previous reading is added to the system clock. When either the kernel or a user process requests the time, the system clock is updated from the additional counter before use.

This scheme has two important advantages over the standard configuration. The granularity of time obtained is now 1 $\mu$s, rather than 10 ms. Losing even a large number of clock interrupts does not affect timekeeping, as the counter is read when time is needed, and the counter only overflows once every 1000 s. Of course, if sufficiently many consecutive clock interrupts were lost that the counter took on the same value twice without being read, time would be lost.

However, experiments showed that the local oscillator driving this clock was inadequate. The frequency of the supplied crystal oscillator was incorrect; this, however, is of little consequence as both standard NTP algorithms and my work correct for this. Of more importance was the lack of stability of the supplied oscillator; it appears that oscillators supplied with different computers of the same model change frequency on the order of 4 ppm over the course of a day when in our office environment.

I then obtained a DEC KWV11-C timekeeping board. This is a Q-BUS module that plugs into MicroVAX or LSI-11 computers, intended for applications that require better timing capabilities than the clocks supplied with those computers. It consists of a 16-bit counter which may be driven by an internal oscillator or externally, and the ability to capture the value of the counter on an external trigger.

I again modified the kernel to use the board for timekeeping, rather than the internal clock. The normal 100 Hz interrupt was still used for scheduling. Because the KWV11-C only had a 16 bit counter and was driven at 1 MHz, only 65.536 ms could be accumulated without ambiguity, but I did not observe loss of time due to overflow of this counter.

The oscillator supplied with this board was substantially better than the one supplied with the computer, but it too exhibited significant variations in frequency. Figure 6.1 shows frequency estimates, predicted offsets, and actual offsets for this oscillator. The top graph shows NTP's frequency estimates as dots, and the estimates made by the `rsadj` technique as lines. While the frequency changes by several tenths of a ppm, it is far more stable than the several ppm excursions seen in the oscillators supplied with typical workstations. Examining `poffset`, ripple due to frequency changes of this local oscillator can be seen, but it is far less severe than for the corresponding graphs of oscillators supplied with workstations (see Figure 4.1 on page 79). Values of actual offsets are small; NTP synchronized the clock well during this time period.

I obtained a low-cost oven-controlled oscillator (OCXO), and attached it to the KWV11-C timekeeping board, so that the OCXO was then used for timekeeping. Because of bandwidth restrictions on the input circuitry, the OCXO was first prescaled to 500 kHz, resulting in 2 $\mu$s precision. Later, I attached other oscillators to the KWV11-C; it provided a very convenient experimental platform for examining various oscillators.

## 6.3 Design of the GDT-TIME board

I desired to attach a high-quality time reference to the computers with which I was experimenting in order to validate my analysis and control methods. Because I was explicitly attempting to do better than had been done before under conditions of high network-induced noise, I could not use measurements over a wide-area network to measure the quality of

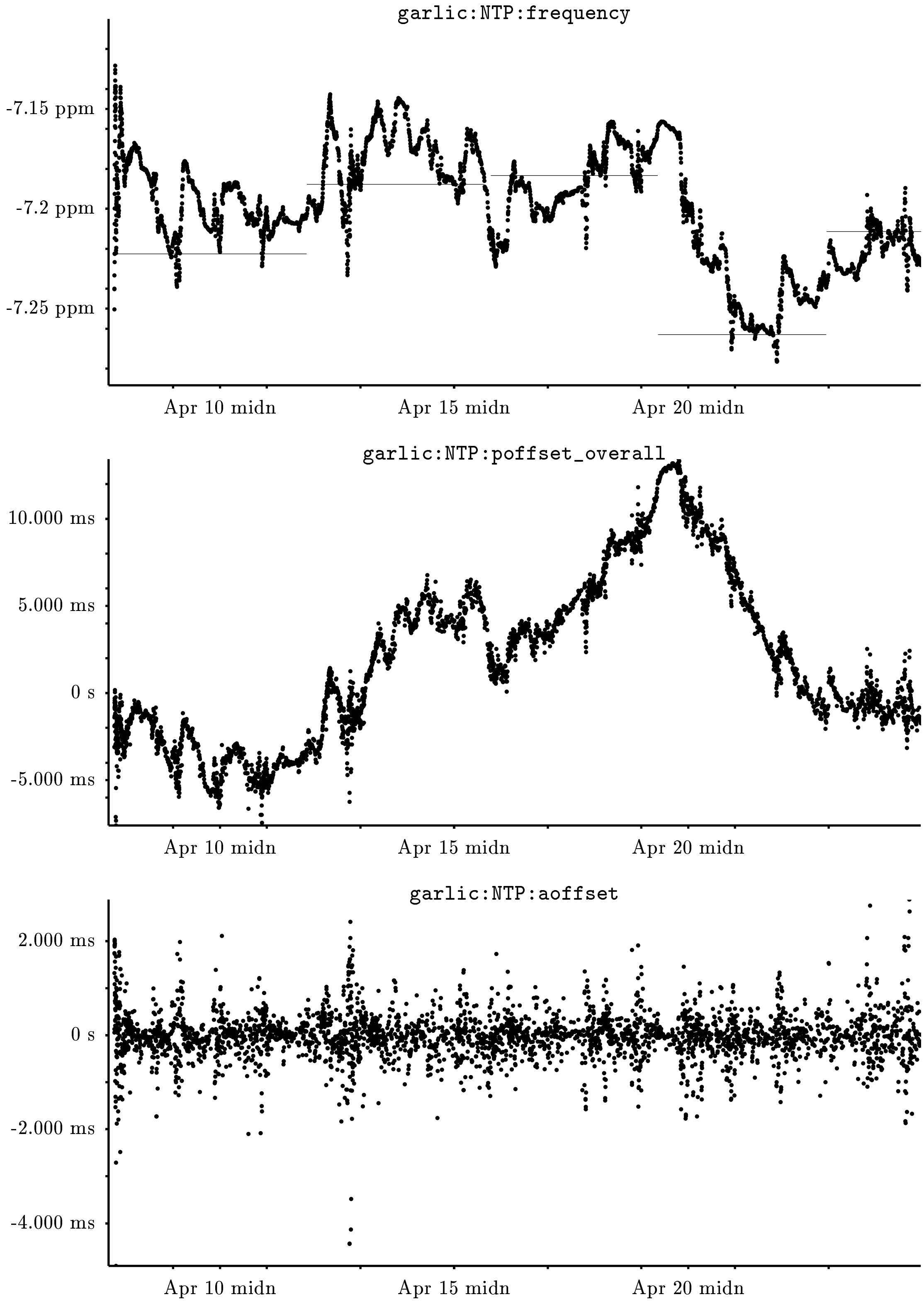



Figure 6.1: Frequency estimates, predicted offsets, and actual offsets of the oscillator supplied with the KWV11-C real-time clock board. Data are from garlic in April of 1992.

synchronization.

I obtained a GPS receiver with a 1 pulse-per-second (PPS) timing output for use as a reference, i.e., a source of `true time` in the terminology of this thesis. Operational considerations dictated placing the GPS antenna on the roof of the building. Given this, I decided to obtain a DEC DS5000/25 (a modern low-cost workstation, also known as a Personal DECStation) for use as the experimental platform. Because the timekeeping facilities furnished with the DECStation were inadequate for my needs, I built a timekeeping board (called the GDT-TIME board) to plug into the DECStation's expansion bus (known as TURBOChannel). The construction of the board was a joint effort between me and Mike Ciholas, my research group's hardware engineer. We had many discussions about the functional specifications for the board, eventually reaching a design which satisfied my needs and was easy to construct. After I prepared functional specifications for the board, Mike drew schematics and programmed the logic array chip. I constructed a prototype, which was used for gathering part of the data in this thesis.

This experience forced me to carefully consider what was desirable in such a device, what was necessary, and what could be omitted. Because of the cooperative design process between myself and a digital designer, and the desire of both of us to produce a minimal but complete solution, I believe I have arrived at a design which does exactly what is required, and not more.

### 6.3.1 Initial design goals

The GDT-TIME board is intended to provide the basic timekeeping functionality required for high-accuracy timekeeping. Beyond providing that functionality, it is intended to be simple and low-cost.

The GDT-TIME board is required to correct several deficiencies of the timekeeping hardware supplied with the DECStation: limited clock resolution, lack of oscillator stability, and lack of ability to capture a timestamp when an external event occurs. It was also designed to solve the problem of lost clock interrupts, and to be able to keep precise time across reboots, and, with optional circuitry and backup power, across power failures.

#### Resolution

The supplied clock has a granularity of 256 Hz, or just under 4 ms. This is on the order of timekeeping errors I was observing with NTP, and greater than the errors I hoped to achieve, and thus clearly inadequate.

I somewhat arbitrarily chose 1 $\mu$s as the required resolution. I had already experimented with a 2 $\mu$s resolution clock, and it was my belief that the limited resolution was not impairing achievable synchronization or my research. It is my opinion that sub-microsecond resolution is not very useful in today's long-haul network environment, and is perhaps marginally useful in the local-area network environment, because it takes longer than a microsecond to read the clock and send a packet. The current Unix timestamp format can represent times with a precision of only 1 $\mu$s, so while the more precise NTP timestamps could be represented, a new interface would be needed to convey the more accurate time to user processes. Given that a system call requires more than than 1 $\mu$s, it did not seem wise to constrain the design to provide greater resolution.

### Oscillator Stability

The local oscillator supplied with the DECStation is not intended for precision timing, and exhibits changes in frequency. I desired that a better oscillator be built into the board, allowing better timekeeping simply by installing a GDT-TIME board. I also desired that it be possible to attach external oscillators to the board for research purposes.

### External event capture

The supplied hardware provides no mechanism for capturing timestamps precisely at external events. I wanted to be able to measure the precise timing of the 1 PPS signal from the GPS receiver with respect to the system clock. Because of my use of the GPS receiver as a source of `true time`, this is actually a measurement of the offset of the system clock with respect to the GPS receiver. Capturing the value of the system clock on the rising edge of an external signal provides the needed functionality.

### Lost interrupts and timekeeping across reboots

I wanted the GDT-TIME board to be resilient against not being read extremely often. This is due in part to the desire to use the board entirely from user code; source code to the Ultrix operating system was not available to me, and therefore I could not modify the kernel to read the board at every clock interrupt as I had done on the MicroVax under BSD.

Additionally, I desired that the board be constructed in such a way that it could be used to keep precision time across reboots. In order for this to be possible, the period of the counter would need to be sufficiently long so that it would be possible to unambiguously compute the current value of time after a reboot. The operating system or timekeeping software would of course need to maintain in some sort of stable storage the correspondence between values of the counter and values of the system clock. With such a scheme, there need be no fundamental difference between reading the counter and updating the system clock once every 10 ms, and reading it 30 minutes later and updating the system clock.

### Timekeeping across power failures

I desired that it be possible to keep precision time across power failures. In theory this is quite simple — simply provide backup power to the critical functionality (that which is required to keep time across reboots), and then power failures can be treated as reboots. Additionally, the counter must have a sufficiently long period to avoid ambiguity over a time period ranging from minutes to weeks, or some other method must be provided to disambiguate the precision time, such as less precise but unambiguous time obtained from the TODR or network.

## 6.3.2 Final design requirements

From the design goals I developed functional specifications for the board. The constructed board is a particular solution for these design requirements, built for the purpose of completing the thesis research.

### Main counter

The board must have a main counter, which is at least 48 bits wide. This counter is to be driven by a nominally 10 MHz on-board oscillator, possibly prescaled to 1 MHz.

If the counter were 48 bits wide, and were driven at 10 MHz, it would overflow and begin counting again after $2^{48}/10$ Mhz = 325.8 days. At 40 bits, the time to wrap is around 31 hours, which should be sufficient for reboots but possibly not for power failures. At 32 bits, the time is just over 7 minutes, which is likely not fast enough to keep time across reboots without additional mechanisms to disambiguate values of the counter. At 64 bits, the time to wrap around is over a million years.

It was a design specification that the processor not be able to set, control or modify the counter in any way. This makes the timekeeping hardware itself robust against crashes and (in cases where power fail protection is present) power failures.

### External capture ability

In order to compare the computer's time to an external reference such as the GPS receiver, the timekeeping board must have one or more external capture inputs. On a rising edge of these inputs, the low-order 32 bits of the main counter is to be captured and stored in a capture register. The processor could then read the capture register and determine the precise time of the rising edge relative to the system clock, which is derived from the main counter.

A capture register size of 32 bits was deemed adequate, since the capture register would normally be checked every few seconds, and there is no need to preserve capture register information across reboots or power failures. I decided that there was no need to have the ability to store multiple capture events, since the purpose of the board is precision timekeeping, not general laboratory use — the intended use is to record the timing of the 1 PPS signal from the GPS receiver, and it is relatively unimportant if an occasional timing mark is lost. It is important, however, that no incorrect values be read.

## 6.3.3 Actual constructed prototype

The actual prototype board constructed and used in the thesis research has the following features:

- 32-bit auxiliary counter driver by TURBOChannel clock (for debugging)
- 64-bit main counter driver by on-board 10 MHz OCXO
- 2 32-bit capture registers driven by external signals

The battery backup scheme to provide precision timekeeping across power failures, while operationally desirable, was not deemed critical to performing the research — with such hardware, power failures and reboots become indistinguishable from a timekeeping point of view.

## 6.4 Summary

Precision timekeeping requires only simple hardware support. A stable local oscillator must be provided, together with a robust mechanism to count the ticks of that oscillator. No ability to set the value of this counter is needed. The ability to capture the value of this counter when an external signal occurs is also helpful.

A stable local oscillator might cost $100 in small quantities; such an expense is not clearly justifiable for a general-purpose computer, as the user might not care about time. However, the cost of including a 64-bit counter with microsecond resolution is negligible, and such a counter should be included in all computers. The ability to capture the value of the counter given an external signal would appear to more costly, as it would seem to require a connector. However, it would be easy to arrange for the value of the counter to be captured when a character arrives on the serial port.

It should also be possible to produce a module similar to the one I constructed for a reasonable price, since the only components with significant cost is the stable oscillator. Given that complex interface modules are routinely sold for less than $100 today, it is clearly possible to produce such a timekeeping module with a stable local oscillator costing $100 for less than $200, given reasonable demand. Persons who care about time synchronization could then purchase such a module and install it in their computers.

# Chapter 7
# Conclusion

In this chapter I summarize the thesis. I also suggest topics for further research, and make suggestions for improving the operational time synchronization system currently in place in the Internet.

## 7.1 Summary

I presented the `rsadj` synchronization measurement technique, and presented and discussed many observations using this technique. I presented the integrated timekeeping scheme, and presented data comparing its performance to that of the the standard NTP algorithms. In the case where the local oscillator was stable to a few parts in $10^8$ and with the conditions found in today's Internet, the integrated timekeeping scheme achieved performance roughly an order of magnitude better than that achieved by NTP.

Both the `rsadj` technique and the integrated timekeeping scheme are the result of taking a fundamentally different view of the nature of synchronizing computer clocks. Rather than adjusting the value of the system clock, observations of remote clocks are recorded relative to the value of the local oscillator, unchanged by the synchronization algorithm. Then, parameters of the local oscillator are estimated. Rather than viewing the clock as something to be set, I treat the local oscillator as an important tool to be used in calculating a value of time based on all available data.

Both the `rsadj` technique and the integrated timekeeping scheme are nonlinear in that they choose to ignore some offset observations completely based on the values of others. This is a powerful method, in that recent observations can be completely ignored if conditions warrant.

I verified both the `rsadj` technique and the integrated timekeeping scheme by using a GPS receiver as a source of true time.

I argued that only simple hardware support is needed for precision timekeeping.

### 7.1.1 The `rsadj` synchronization measurement technique

The `rsadj` synchronization measurement technique separates the behavior of an existing synchronization algorithm from that of the local oscillator, network, and remote clocks. The technique produces views of remote clocks with respect to the local oscillator; such views are uncontaminated by the actions of the synchronization algorithm. It also produces views of the system clock with respect to the local oscillator; this view shows precisely the actions of the synchronization algorithm.

I showed that it is possible to separate the behavior of the local oscillator from that of the network and remote clock by observing multiple remote clocks. Strong similarities in the view of multiple remote clocks with respect to the local oscillator reflect that the features are due to the behavior of the local oscillator. Differences in the views of multiple remote clocks indicate that one or the other remote clock either was observed via a high-variance network or did not always have the right time. By observing several remote clocks, one can determine the stability of the local oscillator, and problematic behavior can be attributed to the local oscillator or to a specific remote clock.

Given a stable local oscillator, one can examine the view of a remote clock with respect to the local oscillator in the form of a wedge plot. Such a plot allows one to determine whether the value of an offset observation can be attributed to delay, or whether it must be attributed to an incorrect remote clock. Observations falling inside the wedge do not indicate that the remote clock had an incorrect value of time, but indicate only that additional delay was encountered. If an offset was in error by more than half the additional delay, the observation would fall outside the wedge. Observations falling outside the wedge indicate that the remote clock had incorrect values of time when these observations were made; they cannot be explained by the additional delay observed.

The `rsadj` technique provides much insight into why NTP does not synchronize clocks better than it does in many cases. I hope that these observations will be useful in the design of future synchronization protocols.

### 7.1.2 The integrated timekeeping scheme

The integrated timekeeping scheme periodically computes estimates of the parameters of the local oscillator, and computes the current time based on those parameters and the value of the local oscillator. It contains mechanisms to cope with differences in systematic offsets among remote clocks. New estimates must be computed sufficiently frequently to avoid errors due to unmodeled frequency changes in the local oscillator. For oscillators sufficiently stable to achieve good results, I found that computing new estimates every one to four hours worked well.

I showed that given an oscillator that is stable to a few parts in $10^8$ and the conditions of today's Internet, integrated timekeeping performs approximately an order of magnitude better than NTP. Given a local oscillator stable to only a few parts in $10^6$, it performed much worse than NTP. However, since the `rsadj` technique indicates that such an oscillator is not stable, the standard NTP algorithms can be run in such situations.

I demonstrated that the integrated timekeeping scheme does not require prohibitively large amounts of resources. While it requires far more CPU time and storage than NTP, my prototype implementation requires less than a fraction of a percent of the available CPU time, and only a few megabytes of storage. I did not attempt to optimize the use of storage by the scheme, and I believe that far less storage is actually required.

While I have shown that the integrated timekeeping scheme has good average-case performance (given a stable local oscillator), I have not made any worst-case performance claims. Such claims are difficult to make; Marzullo and Owicki ([MO85]) describe a method for calculating provable error bounds on values of clocks, but such bounds are far greater than the average case performance achieved by NTP or integrated timekeeping. One limitation to performance is the stability of the local oscillator; in practice almost all local oscillators are more stable than their worst-case specifications. Such characteristics of real systems make it difficult to produce performance guarantees, as it is difficult to obtain performance guarantees from the underlying mechanisms that are valid and are also tight bounds on the real behavior.

### 7.1.3 Hardware support for timekeeping

I showed that timekeeping hardware supplied with many computer systems is inadequate for precision timekeeping. I discussed design issues relating to a hardware timekeeping module constructed for this research. I argued that the core requirement is to reliably accumulate the output of a stable oscillator with sufficient precision. Except for providing a highly stable oscillator, I argued that this functionality could be provided at very little cost in computer systems.

### 7.1.4 Limitations on good timekeeping

In making the observations presented in this thesis, I encountered three classes of limitations to better timekeeping: lack of local oscillator stability, lack of timekeeping accuracy of the remote clock, and network delay variance. The work described in this thesis achieves improved performance, and it is interesting to consider the effects of the three classes of limitations on the schemes presented here.

The `rsadj` technique does not strictly depend upon a stable local oscillator; it can indicate that the local oscillator was not stable. In such cases, the integrated timekeeping scheme performs poorly, and the `rsadj` technique indicates that it performs poorly. The `rsadj` technique is most useful when the local oscillator is sufficiently stable that a single set of estimates can describe observations covering several days or more. With the characteristics of today's Internet, this corresponds to stabilities of a few parts in $10^8$. It is under such conditions that the timekeeping accuracy of remote clocks can be evaluated. Also, with a stable local oscillator and an accurate remote clock, the effects of delay variance can be clearly seen, and information can even be determined about changes in one-way delays.

Given a stable local oscillator, the effects of the other classes of errors can be considered. If the timekeeping of the remote clock is poor, the `rsadj` technique identifies this fact. I intentionally refer to the timekeeping of the remote clock rather than its stability in order to emphasize that observations are made of the remote clock's system clock, rather than of its local oscillator. Because the computation of predicted time attempts to identify and discount observations taken when the remote clock's timekeeping is poor, the integrated timekeeping scheme tolerates such errors reasonably well. However, the necessity of being prepared for the possibility of the remote clock being incorrect hurts the performance of the integrated scheme in systems with poor local oscillators. If the remote clocks were correct with overwhelming probability, the procedure to mark groups of inconsistent observations invalid could be omitted, and the scheme would respond more quickly to changes in the frequency of the local oscillator.

If network delay variance is the major obstacle to precision timekeeping, as it is when observing a well-behaved remote clock with a stable local oscillator, the `rsadj` technique and the integrated timekeeping scheme work extremely well, and result in significantly better synchronization performance than that achieved by the standard NTP algorithms.

If the local oscillator is not stable, then errors due to network delay variance and incorrect behavior of remote clocks become critical. With a low-variance network and a remote clock that is always correct, accurate local time is achievable; NTP in fact achieves good synchronization under this case. Assume for a moment that there exists a self-monitoring scheme where remote clocks would fail to answer requests for time rather than returning erroneous indications. Then, the integrated timekeeping scheme could be modified by removing the step that identifies and marks invalid intervals in which the remote clock appears to behave erroneously. Then, if the variance in network delay is not extremely large, integrated timekeeping should function reasonably well, even with a local oscillator of limited stability.

### 7.1.5 Comments on NTP

Earlier, I made suggestions for improvement to the Network Time Protocol. These suggestions involved small changes to processing steps intended to mitigate the effects of network delay variance and differences in systematic offsets among remote clocks.

NTP as it currently exists is not capable of taking full advantage of the stability of the local oscillator. This is unfortunately inherent in the linear phase-locked loop model adopted to control the local clock. Because the local-clock algorithms are linear, they cannot ignore several hours of data at times and follow it at others. For example, if the last 12 hours of data from what is normally the most stable remote clock has elevated delay values, and the segments covering that time have very high rms errors, a timekeeping scheme would do well to completely ignore that data, and use older data and data from other remote clocks. However, the linear model is not capable of such nonlinear behavior. The integrated timekeeping scheme analyzes the stability of the local clock and the timekeeping accuracy of remote clocks, and then decides which of the data to include in the final calculation of the current time. This nonlinear process is very powerful, as it allows the scheme to completely ignore incorrect data.

In conducting the thesis research, I examined many hundreds of graphs indicating the behavior of NTP in various circumstances. While I have criticized the protocol for lack of high-level accuracy in a limited set of circumstances, I must also point out that the ability of NTP to deliver reasonable time synchronization over a wide range of conditions is most impressive.

### 7.1.6 Use of actual offsets to indicate synchronization quality

David Mills presents graphs of observed offsets, values of `aoffset` in the terminology of this thesis, as a way of showing that good synchronization was achieved ([Mil93], pp. 32–34). While observing only small offsets to a reliable source indicates good timekeeping, such graphs can be very misleading. For example, consider the case of a computer running NTP tracking a remote clock which undergoes a positive phase excursion for several hours. Values of `aoffset` would be more often positive than negative while the system clock's phase was increased, and more often negative than positive while the clock's phase returns to the correct value. If the phase excursion were fairly gradual, such an effect might not be noticeable.

Figure 3.1 on page 50 shows such a situation. Rather than a gradual phase excursion, however, the effect of NTP switching which remote clock primarily controls the value of the system offset causes a sudden phase excursion. Examination of only the graph of actual offsets does not show this effect as clearly as examining values of predicted offsets computed by the `rsadj` technique. Were the excursion more gradual, the few large values of `aoffset` would be smaller, as the NTP local-clock algorithms would have begun to track the excursion, reducing the values of observed offsets.

Thus, I caution researchers to be very careful in interpreting values of observed offsets, and suggest that they also use the `rsadj` technique and examine values of predicted offsets.

## 7.2 Suggestions for further research

In this section I suggest possible improvements to the integrated timekeeping scheme, make suggestions for a study of network delay properties, and discuss possible enhancements to NTP.

### 7.2.1 Improvement of integrated timekeeping scheme

After developing the `rsadj` measurement technique and the integrated timekeeping scheme, I have many ideas for further development and improvement of schemes for time synchronization over packet networks. Some of these ideas are incremental improvements to concepts and procedures already present, and some involve adding new mechanisms.

#### Calculation of provable error bounds

The scheme of Marzullo and Owicki could be applied to the integrated timekeeping scheme in order to calculate bounds on the error of the system clock. This could be done using realistic worst-case assumptions about the components of the system. For example, oscillators might be assumed correct to within 100 ppm, and it might only be assumed that network delays are nonnegative, rather than assuming some degree of symmetry. Perhaps the most troublesome assumption is the accuracy of remote clocks. In the scheme of Marzullo and Owicki, each remote clock gives an error bound when returning an indication of time, and their scheme *preserves* correctness of the error bounds. However, I have observed conditions where remote clocks appear to be incorrect beyond their error bounds; I believe such conditions correspond to failures of radio timecode receiver reference clocks to perform according to their specifications.

Another approach would be to calculate tighter bounds by using assumptions about the components derived from observing them. For example, even the unstable oscillators for which data were presented earlier in this thesis appeared stable to within a range of 3 or 4 ppm. It should be possible to observe an oscillator for several weeks, and then construct assumptions based on that behavior that will be true with overwhelming probability for the next several weeks. For example, an oscillator that has been observed to be stable to within 3 or 4 ppm could be assumed to be stable within 9 to 12 ppm; it is not likely that such a bound would be exceeded.

By estimating a base value of network delay, much tighter bounds on errors due to the network could be calculated, rather than merely half the total observed delay. Such bounds would be in terms of frequency stability, rather than time, because of the impossibility of distinguishing systematic offsets from long-term patterns of asymmetric delays.

Calculating both provable and "very likely" error bounds according to the techniques of Marzullo and Owicki would be a significant improvement to integrated timekeeping. The good average-case performance would be retained, and both true worst-case bounds and bounds that are not exceeded with high probability would be useful to different clients of synchronization.

### Use of wedge concept to compute error estimates

Scatter plots of offset vs. delay have been presented, and have generally had striking wedge shapes. This is to be expected, as additional delay can induce an error of one-half the extra delay. Rather than use rms error to judge how well a set of estimates describes data, it should be possible to compute a measure of how far offset observations fall outside the wedge.

To do this, one would first find the minimum delay, or perhaps the first percentile of delay, and call this $d_{\min}$. Then, rather than computing rms error, let the error for a given sample $(d, o)$ be

$$e = \max\left(0, |o| - \frac{d - d_{\min}}{2}\right)$$

Then, the sum of the squares of the errors $\sum e^2$ can be computed.

### Improved division of observations into small segments

Currently, the `rsadj` technique divides the observations into small segments. There are three separate purposes for this division — estimation of average error due to network delay variance, identifying and removing observations for which the remote clock had incorrect time, and providing a starting place for combining similar segments. It is possible that improved behavior would result from careful consideration of the requirements for each purpose and designing a method of division appropriate for each purpose.

### Integrated timekeeping with poor oscillators and good remote clocks

The integrated timekeeping scheme could likely be tailored to conditions of unstable local oscillators and good networks if the assumption could be made that the remote clocks are correct. In this way, very good synchronization could be obtained for all computers on a local network by equipping several of them with high-quality local oscillators. The computers with high-quality oscillators would then synchronize to remote clocks using the integrated timekeeping scheme over comparatively poor network conditions. The other computers would estimate their short-term local oscillator parameters frequently, and would rapidly adapt to changes in frequency error, because even a few observations of the computers with high-quality local oscillators would accurately indicate the new parameters.

### Intervals used in the integrated computation

Currently, when computing integrated predicted time, offset observations are used for each remote clock according to the last interval computed by the piecewise computation for that clock. While this procedure is computationally simple and easy to describe, it has the unfortunate drawback that more data are used from remote clocks with higher network delay variance. The amount of data used for each remote clock is arguably appropriate if one had data from *only* that clock with which to synchronize. However, given multiple remote clocks, it is likely that better schemes can be devised.

The length of the last interval for a given clock represents the time over which the estimated error due to network delay variance and the estimated error due to unmodeled frequency changes in the local oscillator are roughly balanced, given the assumption that the remote clock is correct. If all remote clocks always had the correct time, then it would perhaps be reasonable to use data only from the smallest interval of all remote clocks. The existence of a remote clock with a short interval would imply that the errors due to network delay variance to that remote clock are small, since errors due to the local oscillator are correspondingly reduced. In such an ideal world, a good course of action would be to weigh the recent observations of that remote clock heavily, and to use observations of the other remote clocks only over that same recent time period.

However, one cannot assume that the remote clocks are always correct. In Chapter 4, I showed how I determined whether a short last interval was due to frequency changes in the local oscillator or in the remote clock. I presented graphs of `poffset` to various remote clocks, and at times argued that they indicated that the local oscillator was not stable, due to correlations in `poffset` of several remote clocks. I argued that they indicated that the remote clock was unstable, due to the fact that `poffset` to other remote clocks was well-behaved. Generally, it was not possible to determine the stability of the local oscillator if only one remote clock could be observed. However, by observing several additional remote clocks, it was possible to decide that the local oscillator was in fact stable and that one remote clock was not. Thus, one might consider a scheme to automate this process.

In order to automatically estimate the quality of the local oscillator, one could compute $E_{ij}$, the rms error of the estimates of the last interval of clock $i$ with respect to the data of the last interval clock $j$. One could also compute $E'_{ij}$, the rms error of the estimates of the last interval of clock $i$ over the data of clock $j$ corresponding to the last interval of $i$. I believe that the values of such matrices would be most helpful in determining which remote clocks are recently well-behaved, and which are not. Then, remote clocks for which longer intervals were computed but are described almost as well by estimates derived from shorter intervals can be identified. For such remote clocks, the additional error introduced by using the frequency estimate from a different remote clock is small compared to the errors due to unmodeled frequency error and network delay variance. Otherwise, the remote clock would not be described "almost as well." While data from such remote clocks are not bad, there is no reason to use the data from the additional past beyond the other shorter intervals, as such data would likely cause the estimates produced from it to be less accurate in the recent past, due to unmodeled frequency errors.

### Clock selection and combining in the integrated scheme

One could attempt to develop algorithms similar in purpose to the NTP clock selection and combining algorithms, but which operated on sets of uncorrected observations rather than the single resulting offset. Such algorithms could be used to determine which clocks to use in the integrated timekeeping estimation, and the weights to be assigned to those clocks. In this way, the experience and theoretical results concerning clock selection and combining could be applied to the integrated timekeeping scheme.

### 7.2.2 Study of asymmetric delays

By equipping computers at two sites with high-quality local oscillators, and gathering data on both of them using the `rsadj` technique, one could later produce not just values of `poffset`, the offsets that would have been observed had the local clock been steered optimally, but values of `ppoffset`, defined as the offsets that would have been observed had *both* clocks been steered optimally. Given such values, one can then assume that *all* non-zero offsets are due to asymmetric variations in delay. Then, one could transform the offset and delay pairs to values of one-way delay for each direction of travel, assuming symmetric base values of one-way delay. Then, the changes in one-way delays could be analyzed.

### 7.2.3 Effects of remote clocks gathering data for the `rsadj` scheme

At the moment one can only make offset measurements to a remote system. While one could consider what would happen if the remote clock used the integrated timekeeping scheme, it is interesting to consider the effects of also making available the remote clock's retrospective estimate of correct time. If the remote clock gathered data sufficient to run the `rsadj` technique, and this data were available, then one could also compute `ppoffset`, the values of observations that would have resulted had both clocks been optimally steered.

Given such values for several remote systems, one could obtain much better information about the performance of the local oscillator. This information could be fed back into an augmented integrated timekeeping scheme that chose the combining limit, for example, based on the more accurate local oscillator stability information. Given values of $\texttt{rsadjresid}_{\texttt{remote}}$, the values of `rsadjresid` for the remote clock, one could base the entire computation of predicted time on values of `ppoffset` rather than `poffset`.

If such information from remote systems were obtained extremely frequently, such a scheme would amount to using the remote clock's local oscillator to transfer measurements made by the remote clock of still further remote clocks or reference clocks. Such a scheme is similar to the generation of synthetic values of capture for garlic based on the offsets observed between ipa and the GPS receiver and between ipa and garlic.

## 7.3 Suggestions for current operational timekeeping

Given the responses of implementations of NTP to timekeeping errors, network delay variance, and systematic offsets, I suggested improvements to the management of the current time synchronization hierarchy currently deployed in the Internet.

Because there appears to be such a great variance between the quality of time of remote clocks that are synchronized to an attached reference clock and of remote clocks that are synchronized via NTP, it may be appropriate for computers attempting to obtain fine synchronization via NTP to be configured to ignore values from remote clocks normally synchronized to reference clocks when such a remote clock is not synchronized to a reference clock. The current NTP suggested algorithms have provisions to address this, but stronger constraints might be helpful.

Sources of systematic offset should be analyzed and removed. Earlier in the course of this research, I thought that there might be large amounts of asymmetric delay in network paths to several remote clocks. Later investigation revealed that the perceived asymmetry in the observations made of one of these remote clocks was due to processing delays at the remote clock. Examination of the network path to the other remote clock with apparent asymmetric delay revealed that it is in fact symmetric. Regardless of the causes of perceived asymmetry, I believe a process of removing actual systematic biases for computers with attached reference clocks by calibration, and then avoiding synchronization over paths with significant asymmetric delay is likely to reduce greatly the perceived differences in values of systematic offset among remote clocks.

# Bibliography


[ABC+89] John E. Abate, Edgar W. Butterline, Roy A. Carley, Paul Greendyk, Adolfo Montenegro, Christopher D. Near, Steven H. Richman, and George P. Zampetti. AT&T's new approach to synchronization of telecommunication networks. *IEEE Communications Magazine*, pages 35–45, April 1989.

[BH45] Charles B. Breed and George L. Hosmer. *Elementary Surveying*. John Wiley and Sons, 1945.

[BHB62] Charles B. Breed, George L. Hosmer, and Alexander J. Bone. *Higher Surveying*. John Wiley and Sons, 1962.

[Bow84] Nathaniel Bowditch. *American Practical Navigator*. Defense Mapping Agency Hydrographic/Topographic Center, 1984.

[GB85] Edward A. Gerber and Arthur Ballato, editors. *Precision Frequency Control*. Academic Press, 1985.

[Lei90] Alfred Leick. *GPS Satellite Surveying*. John Wiley and Sons, 1990.

[LSW91] Barbara Liskov, Liuba Shrira, and John Wroclawski. Efficient at-most-once messages based on synchronized clocks. *ACM Transactions on Computer Systems*, 9(2):125–142, May 1991.

[Mil91a] David L. Mills. Internet time synchronization: The Network Time Protocol. *IEEE Transactions on Commmunications*, 39(10):1482–1493, October 1991.

[Mil91b] David L. Mills. On synchronizing dartnet clocks to the millisecond — a report. Technical report, University of Delaware, February 1991.

[Mil92a] David L. Mills. Modelling and analysis of computer network clocks. Technical Report 92–5–2, University of Delaware, May 1992.

[Mil92b] David L. Mills. RFC–1305: Network Time Protocol (version 3): Specification, implementation and analysis, March 1992.

[Mil93] David L. Mills. Precision synchronization of computer network clocks. Technical Report 93–11–1, University of Delaware, November 1993.

[Mil94] David L. Mills. RFC–1589: A kernel model for precision timekeeping, March 1994.

[MO85] Keith Marzullo and Susan Owicki. Maintaining the time in a distributed system. *ACM Operating Systems Review*, 19(3):44–54, July 1985.

[Par83] Benjamin Parzen. *Design of Crystal and Other Harmonic Oscillators*. John Wiley and Sons, 1983.

[RLT50] Harry Rubey, George Edward Lommel, and Marion Wesley Todd. *Engineering Surveys: Elementary and Applied.* The Macmillan Company, 1950.

[WADL92] Marc A. Weiss, David W. Allan, Dick D. Davis, and Judah Levine. Smart clock: A new time. *IEEE Transactions on Instrumentation and Measurement*, 41(6):915–918, December 1992.

[Wil90] D. R. Wilcox. Backplane bus distributed realtime clock synchronization. Technical Report 1400, Naval Ocean Systems Center, December 1990.