NTP Test Plan

Sachin Kamboj, David L. Mills

December 20, 2005

1 Aim

This document describes a plan for testing whether an implementation of NTP conforms to the NTP protocol specification. It is meant to complement the NTP protocol specification and hopes to help an implementor of the NTP protocol determine whether or not his/her implementation will interoperate with other NTP implementations.

This document is meant to be necessary but NOT sufficient or complete. By necessary, we mean that any implementation of NTP must be able to succeed in these tests. By not being sufficient or complete, we mean that not all the tests required to test for conformance have been provided here. In particular, only the on-the-wire protocol is being tested here — we do not describe tests for the assortment of algorithms required to ensure that the clocks keep ticking with accuracies in the range of milliseconds or higher. Also we do not describe every single permutation and combination of NTP modes and configurations possible. Hence, we shy away from saying that an implementation that passes these tests is "certified" to conform to the NTP protocol. These tests are only meant to be a guideline and should be supplemented with additional tests as required.

2 Scope

There tests are centered around a single host machine running the NTP implementation being tested and are designed to exercise the following NTP modes of operation:

1. **Client/Server Modes:** These are the most common NTP modes of operation on the Internet today. In these modes, a host that requires timing information (the client) sends a request to a designated unicast server and expects a reply from that server. In some contexts this would be described as a "pull" operation, in that the client pulls the time and proventic values from the server.

2. **Symmetric Modes:** Symmetric modes or *peer* modes are intended for configurations were a clique of low-stratum peers operate as mutual backups for each other. Each peer operates with one or more primary reference sources, such as a radio clock, or a subset of secondary servers known to be reliable and proventicated. Should one of the peers lose all reference sources or simply cease operation, the other peers will automatically reconfigure so that time and proventication values can flow from the surviving peers to all the others in the

clique. In some contexts this would be described as a "push-pull" operation, in that the peer either pulls or pushes the time and proventic values depending on the particular configuration.

Note that symmetric peers operate with their sources in some NTP mode and with each other in symmetric mode.

3. **Broadcast Modes:** Broadcast modes are intended for configurations involving one or a few servers and a possibly very large client population on the same subnet. In broadcast modes, the broadcast server should generate broadcast messages continuously at some pre-specified interval. A broadcast client should respond to the first message received, after waiting for a random interval, by polling the server in burst mode in order to quickly set the host clock and validate the source. Once the clock has been set, the broadcast client should not send any further messages to the server and should simply listen for broadcast messages on the designated interface.

We test all of the above modes both with and without authentication. We only describe tests for symmetric key authentication and leave the tests for the autokey protocol for a subsequent document.

We also do not cover the multicast, anycast and orphan modes. Please refer to the NTP protocol specification for more information on these modes of operation.

3 Testing Methodologies

Some of these tests use the contents of the "over-the-wire" NTP packets to test for conformance to the packet format given in the protocol specification. Other tests require some mechanism to test the value of certain state variables that are specified in the protocol specification. In the reference implementation, the value of these state variables can be checked by using the *ntpq* program included with the reference implementation.[1]

In some tests for interoperability, we require the use of a conformant implementation and test the interoperability between the conformant implementation and the implementation being tested. These tests also require some mechanism for testing the state variables for the peer in the conformant implementation. To avoid the chicken and egg problem of how to test the conformant implementation for conformance, we assume that the reference implementation is conformant and has been tested to be conformant by some external means.

In all of these tests, we explicitly state whether a host is running the conformant implementation or the implementation being tested (henceforth called the *'test implementation'*).

4 Test Configurations

These tests require the hosts being used in the tests to be configured in various modes with various optional parameters being used to modify the modes of operation. To

[1] We do not require implementations to support *ntpq* or to provide a similar test program. We simply require some mechanism for verifying that the state variables being tested have the desired values.

allow the exact mode of operation to be specified in a succinct way, we provide a configuration file for each of the hosts being used in the tests. The format of the configuration file is based on the *ntp.conf* configuration file used in the reference implementation. In this section, we briefly describe the meanings of the various commands used in the configuration file. Note that implementations are not required to support this configuration format — they are simply required to provide some mechanism for setting and controlling the mode of operation of the hosts and hence are free to implement their own configuration options.

4.1 Specifying the mode of operation

We specify the mode of operation of a host using a command of the form:

<mode><address><options>

where mode is one of the the following:

1. **server:** This host should be configured to be a **client**, that is, the host should send regular NTP mode 3 packets (client packets) to the specified unicast *<address>* (server address). A compliant server is required to reply to such messages in accordance with the NTP protocol specification.

2. **peer:** This host should be configured in **symmetric active mode**, that is, NTP mode 1. The host being configured should mobilize a permanent association for the specified peer. In this mode the local clock can be synchronized to the remote peer or the remote peer can be synchronized to the local clock, depending on which peer is the better source of time (the peer that has the shorter root synchronization distance).

3. **broadcast:** This host should be configured to be a broadcast server. This host should send our regular broadcast messages at some pre-specified interval to the broadcast *<address>* specified. Note that the NTP protocol specification recommends that a broadcast server support normal unicast messages and respond to them as would a normal NTP server as this allows the broadcast clients to calculate the propagation delay to the broadcast server.

4. **broadcastclient:** This host should be configured to be a broadcast client and should listen for broadcast messages on the designated broadcast *<address>*. Such a host should not normally respond to any broadcast messages. However, a host may choose to respond to the first broadcast message received by sending out a volley of eight messages at 2 second intervals, after waiting for a random time interval. This allows a broadcast client to authenticate the server and calculate the propagation delay.

Note that the *ntp.conf* configuration file in the reference implementation does not include any command for explicitly configuring a host to act as an NTP server or for configuring a host to work in symmetric passive peer mode, nor is such a command required. Instead, a host acts as a server if it receives a client packet (NTP mode 3 packet) from a host and if it supports the server mode of operation. Similarly, a host acts as a symmetric passive peer if it receives a symmetric active request from

another host (an NTP mode 1 packet) and if it is capable of supporting the peer modes of operation. Hence, these two modes of operation are entered implicitly, and there is no way to "force" a host to work in these modes.

Also note that the server mode differs from the symmetric passive mode in the way in which it handles requests from the client/symmetric active host. In the server mode of operation, a server should not mobilize an association for handling the client request. Instead, it should service the client's request without saving any persistent state for the client.

A host operating in symmetric passive mode, on the other hand, should mobilize an ephemeral connection for the symmetric active peer. Once the association has been mobilized, the two peers (the symmetric active and passive hosts) should be indistinguishable — that is each peer should maintain the same state information for the other host. The only difference between a persistent association and an ephemeral association is the fact than an ephemeral association may be demobilized due to timeout or an error condition.

4.2 Using Authentication

Authentication is required, by default, in certain modes, for example, in the symmetric modes and the broadcast modes and is optional but recommended for the other modes of operation. NTP offers two authentication schemes: the symmetric key authentication mechanism described in RFC 1305 and the autokey protocol described in *Public-key cryptography for the Network Time Protocol*. Network Working Group Request for Comments RFC-XXXX, University of Delaware, August 2003.

We do not include tests for the autokey protocol in this document and leave that for a future document.

To use symmetric key authentication, first the relevant keys need to be generated and distributed to the computers involved in the exchange. In the reference implementation, a set of symmetric keys can be generated by giving the following command at the command line:

ntp-keygen -M

This command rolls out a set of MD5 keys that can be used to generate a hash of the data in the NTP packets. Note that the NTP data is still public and is not encrypted in any way. The generated keys are stored in a file by the name

```
ntpkey_MD5key_<hostname>.<timestamp>
```

where hostname is the name of the host where the keys were generated and timestamp is the local time at which the keys were generated. The file itself looks like this:

```
# ntpkey_MD5key_ferrari.local.3340517626
# Wed Nov  9 04:33:46 2005
 1 MD5  /^xwuKB_mf[{Hup # MD5 key
 2 MD5  $oQogYnhxW^MrQ& # MD5 key
 3 MD5  lV<<Cc"gk0)ll<c # MD5 key
```

```
 4 MD5  LAU\]_$;;-[;}5O # MD5 key
 5 MD5  CiWPKOpj<jse@)y # MD5 key
 6 MD5  S6bz\H!0'9"m!u" # MD5 key
 7 MD5  [Rp?2MGnjoA9>{* # MD5 key
 8 MD5  K;{8P-]y)g6Ul~r # MD5 key
 9 MD5  ()"M/iPv@ddC@@s # MD5 key
10 MD5  =L\m&96wx6tee'n # MD5 key
11 MD5  2eg'a<'VD[dx?zf # MD5 key
12 MD5  }]\Cbgk!^$Fd(af # MD5 key
13 MD5  E0v&_pnUnrQArAZ # MD5 key
14 MD5  Sz|wd[X1-Nfo[;Y # MD5 key
15 MD5  yQ7_>zpb\|=x0qu # MD5 key
16 MD5  X|"Sf)vR>f$r6=r # MD5 key
```

After the keys have been generated, the keys have to be distributed to all the computers that will participate in the authentication exchange. Also some mechanism is needed to point the implementation to the correct set of keys to be used for the exchange. In the reference implementation, this can be done by giving the following commands in the *ntp.conf* configuration file:

```
keys <path-to-keys file>
trustedkey <key_ids that are valid/trusted>
```

We also use the above commands in the configuration files given for the various tests to indicate that symmetric keys need to be loaded from the specified file and which symmetric keys are trusted. An implementation is free to use its own format for the symmetric keys file and for indicating which keys are to be trusted. To allow interoperation between the implementation being tested and the reference implementation (as is required by these tests), we do require some mechanism for converting the symmetric keys from the format used in the implementation being tested and the reference implementation.

When authentication is being used, we require some mechanism to make this explicit in the configuration file. Again we follow the reference implementation. The reference implementation appends the option *key <key_id>* to the command indicating the mode of operation to indicate that symmetric key authentication is being used and that *<key_id>* is the key to be used.

For example, the command:

```
keys /foo/bar
trustedkey 3,4,5,6
server T key 5
```

indicates that the host should load the keys from a file named /foo/bar, that the keys 3,4,5 and 6 are trusted, that the host is operating in client mode using a symmetric-key authenticated server T and that the key with key-id 5 is to be used for performing the authentication.

5 Synchronization

We assume that a host is synchronized to a peer the following two conditions are satisfied:

1. **The host receives at least one *"clean"* packet from the peer for the last eight packets sent.** We define a *"clean"* packet as one which satisfies the following tests, as specified in the protocol implementation:

   Test 1: The packet received must NOT be a duplicate packet. To check for duplicate packets, an implementation must store the transmit timestamp on the last clean packet that was received. Then to determine if a newly arrived packet is clean, the implementation must ensure that the transmit timestamp on that packet is different from the stored timestamp.

   Test 2: The packet received must NOT be a bogus packet. To determine bogus packets, the implementation must ensure that the origin timestamp of the packet is the same as that of the last packet sent by the host to this peer. This test only applies in client/server and symmetric modes.

   Test 3 and Test 6: The peer must itself be synchronized. To test for this, the following conditions must hold:

   (a) The origin timestamp of the packet must not be zero. This test only applies in client/server and symmetric modes.

   (b) The leap bits must not be 11.

   (c) The stratum of the peer must be within a certain pre-specified range.

   Test 4: If access control is used, the peer must be allowed access to the host. Access control is not a feature of the NTP protocol itself and is not a part of the protocol specification. However, implementations are free to implement some mechanism for allowing only hosts from a particular subnet or hosts with a specified IP address to serve as the sources of time. Access control may be used by servers to restrict the set of clients that it provides service to or it can be used by clients to prevent clogging attacks or packets from untrusted peers. Note that access control should not be used as an alternative to authentication as source address based restrictions are easily circumvented by a determined cracker.

   Test 5: If authentication is used or required, the packet must be correctly authenticated. See Section 4.2 for more information on authentication.

   Test 7: The packet header must pass the following sanity tests:

   (a) The delay (δ) should not be negative.

   (b) The dispersion (ϵ) should not be negative.

   (c) The ratio $\frac{\delta}{2} + \epsilon$ should be less than the pre-specified maximum dispersion.

   (d) The transmit timestamp should be greater than the reference tiimestamp.

   Note that additional tests, which are omitted here, are required if the autokey protocol is being used. In the reference implementation, the result of running

each of these tests can be verified by looking at the flash bits. See Appendix B for more details.

2. **The root distance falls below a predetermined threshold.** The root distance is defined as the sum of half the delay to the root and the dispersion of the host.

6 Tests

Test ID: NTP.1

Test Aim: To test that a NTP client can synchronize to an external synchronization source.

Modes being tested: Client

Security Protocol being tested: None

Notes: See Section 5 for more information about when a client is assumed to be synchronized.

Requirements:

1. A synchronized source of time, T, running the conformant implementation.
2. A client host, C, running the implementation to be tested.

Configuration Files:

1. C:
   - ntp.conf:
     server T

Approach and Expected Results:

1. Startup the client, C.
2. Use 'tcpdump', 'ethereal', or a similar packet monitoring program to verify that the header contents are as specified in the NTP protocol specification.
3. Verify that the client synchronizes to T as described in Section 5 — that is, the client, C should receive at least one "clean" packet from T and the synchronization distance should fall below the pre-specified threshold.

Test ID: NTP.2

Test Aim: To test the operation of a host running as an NTP server.

Modes being tested: Server

Security Protocol being tested: None

Notes: In this test, we wish to see if a host configured as an NTP server works according to the specification. In particular, the test host, S, should accept NTP packets, process them and reply to them.

Requirements:

1. A synchronized source of time, T, running the conformant implementation.
2. A server host, S, running the implementation to be tested.
3. A client host, C, running the conformant implementation.

Configuration Files:

1. S:
   - ntp.conf:
     server T
2. C:
   - ntp.conf:
     server S

Approach and Expected Results:

1. Startup the server S and confirm that it synchronizes to T as explained in the Test 1.
2. Startup 'tcpdump', 'ethereal', or a similar packet monitoring program on host C to verify the header contents.
3. Startup host C and verify that for each NTP request from client C (NTP mode 3 packet), server S sends a response packet (NTP mode 4).
4. Also verify that:
   (a) C synchronizes to S as specified in Section 5
   (b) The mode of the response packets is 4.
   (c) The value of the originate timestamp in S's reply is the same as the value of the transmit timestamp in C's request message.
   (d) The transmit time is greater than the receive time.
   (e) The stratum of S is one greater than the stratum of T.
   (f) The reference identifier for S is the source of synchronization, T.
5. Run 'ntpq' on host C and use the 'pstatus' command to verify that peer variables are set to reasonable values as indicated in the NTP protocol specification.

Test ID: NTP.3

Test Aim: To verify the operation of a host in client/server mode that looses its source of synchronization

Modes being tested: Client/Server

Security Protocol being tested: None

Notes: When a synchronization source becomes unreachable, the client host will send regular packets to the server but receive no reply. The client should mark the server as being unreachable after it has not received any reply for the last eight consecutive packets sent.

Requirements:

1. A synchronized source of time, T, running the conformant implementation.
2. A server host, S, running the conformant implementation.
3. A client host, C, running the implementation being tested.

Configuration Files:

1. S:
   - ntp.conf:
     server T
2. C:
   - ntp.conf:
     server S

Approach and Expected Results:

1. Startup hosts S and C and verify that S synchronizes to T and C to S.
2. Kill the NTP program on host S and wait for approximately 10 NTP packets to be sent by host C.
3. Use some mechanism to verify that C recognizes that S is unreachable and that it no longer displays S as its source of synchronization.

Test ID: NTP.4

Test Aims:

- To verify that the poll interval increases after the host first synchronizes to a server.
- To verify that the poll interval increases when a host looses its source of synchronization.

Modes being tested: Client/Server

Security Protocol being tested: None

Notes: As specified in the NTP protocol specification, the poll interval should increase from *minpoll* to *maxpoll* both after the host successfully synchronizes to its source of synchronization and also when the host looses its source of synchronization. This is done in order to reduce the network traffic. In the reference implementation, the poll interval increases from 64s to 1024s in both these cases.

Requirements:

1. A synchronized source of time, T, running the conformant implementation.
2. A server host, S, running the conformant implementation.
3. A client host, C, running the implementation being tested.

Configuration Files:

1. S:
   - ntp.conf:
     server T
2. C:
   - ntp.conf:
     server S

Approach and Expected Results:

1. Startup hosts S and C and verify that S synchronizes to T and C to S.
2. Use some mechanism to verify that the poll interval increases to *maxpoll* over a period of a couple of hours for both S and C. This can be verified by queering the value of the peer.poll variable in the reference implementation using the ntpq program. To do this, run the command 'peers' at the ntpq command prompt and check the value under the poll column.
3. Kill the ntp programs on both S and C. Now restart NTP on both S and C and verify that S synchronizes to T and C to S as in Step 1. After a couple of minutes, kill the NTP program on host S and check the value of C's poll interval over the next hour. The poll interval should gradually increase to *maxpoll*.

Test ID: NTP.5

Test Aim: To verify the operation of a client host that is using symmetric key cryptography

Modes being tested: Client

Security Protocol being tested: Symmetric/Private Key

Notes: None

Requirements:

1. A synchronized source of time, T, running the conformant implementation.
2. A server host, S, running the conformant implementation.
3. A client host, C, running the implementation being tested.

Configuration Files:

1. S:

- ntp.conf:

```
keys <path-to-keys file>
trustedkey 5
server T
```

2. C:
   - ntp.conf:

```
keys <path-to-keys file>
trustedkey 5
server S key 5
```

Approach and Expected Results:

1. Generate and distribute the symmetric keys as explained in Section 4.2.
2. Start S and C at the same time and verify that S synchronizes to T within five minutes and that during this time C does not synchronize to S.
3. Verify that after ten minutes C does synchronize to S.
4. Stop C and modify the symmetric key being used in C in some way so that S and C no longer share the same symmetric keys.
5. Restart C with the updated symmetric keys and verify that C no longer synchronizes to S. Also verify that after a few minutes, C starts to back off its poll interval to *maxpoll* seconds.

Test ID: NTP.6

Test Aim: To verify the operation of a server that is providing time to clients that are using symmetric key cryptography to authenticate the server.

Modes being tested: Server

Security Protocol being tested: Symmetric/Private Key

Notes: None

Requirements:

1. A synchronized source of time, T, running the conformant implementation.
2. A server host, S, running the implementation being tested.
3. A client host, C, running the conformant implementation.

Configuration Files:

1. S:
   - ntp.conf:

```
keys <path-to-keys file>
trustedkey 5
server T
```

2. C:
   - ntp.conf:

```
keys <path-to-keys file>
trustedkey 5
server S key 5
```

Approach and Expected Results:

1. Generate and distribute the symmetric keys as explained in Section 4.2.
2. Start S and C at the same time and verify that S synchronizes to T within five minutes and that during this time C shows refid INIT and does not synchronize to S. (Run the 'ntpq' program on host C to verify this. Give the command 'peers' at the ntpq command prompt and verify that the refid field shows INIT for the peer S.)
3. Verify that after ten minutes C does synchronize to S. Again the 'ntpq' program can be used to verify this. See Appendix A for more details on how to check for this.
4. Stop S and modify the symmetric keys being used in S in some way so that S and C no longer share the same symmetric keys.
5. Restart S with the updated symmetric keys and verify that C no longer synchronizes to S. Also verify that the refid for S on C shows NKEY and the syslog shows an appropriate message. Verify after a few minutes C starts to back off its poll interval to 1024 s.

Test ID: NTP.7

Test Aim: To verify the operation of two host in peer modes

Modes being tested: Symmetric Active/Symmetric Passive

Security Protocol being tested: Symmetric

Notes:

1. By default authentication is needed for symmetric modes.
2. The host initiating the association is working in symmetric active mode while the other host is working in symmetric passive mode.
3. The symmetric active host mobilizes a persistent connection while the symmetric passive mode mobilizes an ephemeral connection.
4. Except for the above two differences, the two hosts operate in the exact same fashion once the associations have been mobilized.

Requirements:

1. Two synchronized sources of time at the same stratum, T and U, running the conformant implementation.
2. A host A, which will become the symmetric passive client, running the implementation being tested.
3. A host B, which will be the symmetric active client, which will also be running the implementation being tested.

Configuration Files:

1. A:
   - ntp.conf:

     ```
     keys <path-to-keys file>
     trustedkey 5
     server T
     ```
2. B:
   - ntp.conf:

     ```
     keys <path-to-keys file>
     trustedkey 5
     server U
     peer A key 5
     ```

Approach and Expected Results:

1. Roll out the symmetric keys as explained in Section 4.2
2. Startup hosts A and B and verify that they both synchronize to their respective servers T and U as explained in Section 5. Wait for a couple of minutes for things to settle down.
3. Use some mechanism to verify that associations are properly mobilized in both A and B and that both the peers show each other as reachable.
4. Now stop B and verify that the association for B in A is properly demobilized some time. Now restart B and verify that A creates a new mobilization for B after some time.
5. Now stop and restart A and verify that the mobilization for A in B recovers from the abrupt restart.

Test ID: NTP.8

Test Aims: To verify the operation of a host in broadcast server mode.

Modes being tested: Broadcast Server

Security Protocol being tested: Symmetric

Notes:

1. By default authentication is needed for broadcast modes.
2. The broadcast server should send out regular NTP mode 5 packets, on the broadcast interface, at the rate of one packet after every *minpoll* interval. In addition, it is recommended that a broadcast server also support normal unicast messages (NTP mode 3 packets) from broadcast clients so that clients may compute the propagation delay and offset to the broadcast server.

Requirements:

1. A synchronized source of time, T, running the conformant implementation.
2. A broadcast server S, running the implementation being tested.
3. A broadcast client C, running the conformant implementation.

Configuration Files:

1. S:
   - ntp.conf:

     ```
     keys <path-to-keys file>
     trustedkey 5
     server T
     broadcast <broadcast-interface> key 5
     ```
2. C:
   - ntp.conf:

     ```
     keys <path-to-keys file>
     trustedkey 5
     broadcastclient
     ```

Approach and Expected Results:

1. Make sure that both S and C are using the same symmetric keys. See Section 4.2 for more information on how to roll out and distribute symmetric keys.
2. Startup S and verify that it synchronizes to T.
3. Verify that S sends NTP mode 5 packets at regular intervals of *minpoll* seconds. Use 'tcpdump', 'ethereal', or a similar packet monitoring software to verify that the packet contents are as specified in the NTP protocol specifications.
4. Startup C and verify that:
   (a) C sends a volley of eight messages in client mode on receiving the first ntp broadcast message from S and that S replies to every one of the unicast messages in server mode.
   (b) C synchronizes to S as evident by the reach register and the flag bits. (Use 'ntpq' to verify this.)

Test ID: NTP.9

Test Aims: To verify the operation of a host in broadcast client mode.

Modes being tested: Broadcast Client

Security Protocol being tested: Symmetric

Notes:

1. By default authentication is needed for broadcast modes.
2. The broadcast client listens for ntp packets on its broadcast interface. On receiving a NTP packet from a previously unknown broadcast server, the broadcast client mobilizes an ephemeral connection and sends a volley of eight messages to the broadcast server to determine the delay and synchronization distance to the server. After the initial volley, the broadcast client simply waits for packets from the broadcast server.

Requirements:

1. A synchronized source of time, T, running the conformant implementation.
2. A broadcast server S, which sends periodic ntp packets on the broadcast interface. S should also be running the conformant implementation.
3. A broadcast client C, running the implementation being tested.

Configuration Files:

1. S:
   - ntp.conf:

     ```
     keys <path-to-keys file>
     trustedkey 5
     server T
     broadcast <broadcast-interface> key 5
     ```
2. C:
   - ntp.conf:

     ```
     keys <path-to-keys file>
     trustedkey 5
     broadcastclient
     ```

Approach and Expected Results:

1. Make sure that both S and C are using the same symmetric keys. See Section 4.2 for more information on how to roll out and distribute symmetric keys.
2. Startup S and verify that it synchronizes to T.
3. Startup C and verify that:

(a) C sends a volley of eight messages in client-server mode on receiving the first ntp message from S.

(b) C synchronizes to S as explained in Section 5

4. Stop C and modify the symmetric key being used in C in some way so that S and C no longer share the same symmetric keys.
5. Restart C with the updated symmetric keys and verify that C no longer mobilizes an ephemeral connection for S and that C is no longer synchronized to any time source.

A Using *'ntpq'*

In the following two sections, we assume that a host is trying to synchronize to one or more peers and we describe how *'ntpq'* can be used to determine the status of the host and the status of the peers as viewed by the host. We, also, assume that the hosts and the peer are currently running and that *'ntpq'* has already been started.

A.1 Determining the status of the peers

To determine the peers that a host is synchronizing to, give the command *'peers'* at the *'ntpq'* prompt. The output should be similar to the one shown below:

```
ntpq> peers

     remote           refid      st t when poll reach   delay   offset  jitter
==============================================================================
+louie.udel.edu  128.4.1.1        2 u  116  128  377   17.950    0.173   4.935
-otc2.psu.edu    128.118.25.5     2 u   42  128  377   30.087   -7.257  48.245
+caesar.cs.wisc. 128.105.201.11   2 u  107  128  377   53.167   -2.665   2.060
 titan.cais.rnp. 196.34.207.255   2 u   24  128  377  193.770   -4.620  10.418
+ns2.pulsation.f 212.37.192.31    3 u   27  128  377  104.172   -2.626  14.175
+80-28-46-78.ads 150.214.94.5     2 u   31  128  377  171.108   -2.580  10.040
*gatekeeper.no-s .GPS.           1 u   46  128  377   95.369   -0.607  14.203
```

The * shows the current source of synchronization, i.e. the peer being currently synchronized to. The *remote* column shows the hostname of the remote peer, while *refid* shows the IP address of the hosts from which the peers derive their source of synchronization. (The contents of this column are the same as the contents of the *refid* field of an NTP received from the peer.)

To view more information about any one peer, we will need the *association id* of that peer. The association ids of all the peers can be obtained using the *associations* command. The output will be similar to the one shown below:

```
ntpq> associations

ind assID status  conf reach auth condition  last_event cnt
===========================================================
  1 26244  94f4   yes   yes  none  candidat   reachable 15
  2 26245  93f4   yes   yes  none   outlyer   reachable 15
```

```
 3 26246  94f4   yes   yes  none  candidat   reachable 15
 4 26247  90f4   yes   yes  none    reject   reachable 15
 5 26248  94f4   yes   yes  none  candidat   reachable 15
 6 26249  94f4   yes   yes  none  candidat   reachable 15
 7 26250  96f4   yes   yes  none  sys.peer   reachable 15
```

Now to view the status of any one of the peers, give the command *'pstatus <assID>'*, where *<assID>* is the association id of the peer whose status is being determined. The output will be similar to the one given below:

```
ntpq> pstatus 26244
assID=26244 status=94f4 reach, conf, sel_candidat, 15 events, event_reach,
srcadr=louie.udel.edu, srcport=123, dstadr=192.168.0.3, dstport=123,
leap=00, stratum=2, precision=-22, rootdelay=0.412,
rootdispersion=38.605, refid=128.4.1.1, reach=377, unreach=0, hmode=3,
pmode=4, hpoll=7, ppoll=7, flash=00 ok, keyid=0, ttl=0, offset=0.173,
delay=17.950, dispersion=7.554, jitter=4.979,
reftime=c71cb3bc.9871c4f6  Wed, Nov  9 2005 12:43:24.595,
org=c71cb59c.b86a845c  Wed, Nov  9 2005 12:51:24.720,
rec=c71cb59c.bba0620a  Wed, Nov  9 2005 12:51:24.732,
xmt=c71cb59c.b60029f1  Wed, Nov  9 2005 12:51:24.710,
filtdelay=    21.95   19.86   20.97   21.82   19.68   17.95   25.01   39.95,
filtoffset=   -1.56    0.32   -2.83   -2.66   -1.50    0.17    0.24   12.45,
filtdisp=      0.00    1.92    3.87    5.81    7.73    9.63   11.55   13.47
```

A.2 Determining the status of a host

To determine the status of a host, give the command *'readvar'* at the *'ntpq'* command prompt. The output is similar to the following:

```
ntpq> readvar
assID=0 status=46f4 leap_add_sec, sync_ntp, 15 events, event_peer/strat_chg,
version="ntpd 4.2.0a@1.1190-r Thu Jul  7 20:53:26 EDT 2005 (1)"?,
processor="i686", system="Linux/2.6.9-gentoo-r1", leap=01, stratum=2,
precision=-20, rootdelay=95.342, rootdispersion=20.180, peer=26250,
refid=64.142.103.194,
reftime=c71cb9e9.bd72bcb5  Wed, Nov  9 2005 13:09:45.740, poll=7,
clock=0xc71cba4b.0c45ed4a, state=4, offset=0.482, frequency=-2.836,
noise=5.826, jitter=9.424, stability=103.553
ntpq>
```

B Flash Code

The flash code is a valuable debugging aid displayed in the peer variables list. It shows the results of the original sanity checks defined in the NTP specification RFC-1305 and additional ones added in NTPv4. There are 13 tests designated TEST1 through TEST13. The tests are performed in a certain order designed to gain maximum diagnostic information while protecting against accidental or malicious errors. The

flash variable is initialized to zero as each packet is received. If after each set of tests one or more bits are set, the packet is discarded.

Tests TEST1 through TEST3 check the packet timestamps from which the offset and delay are calculated. If any bits are set, the packet is discarded; otherwise, the packet header variables are saved. TEST4 and TEST5 are associated with access control and cryptographic authentication. If any bits are set, the packet is discarded immediately with nothing changed.

Tests TEST6 and TEST7 check the health of the server — If any bits are set, the packet is discarded; otherwise, the offset and delay relative to the server are calculated and saved.

TEST8 and TEST9 check the authentication state using Autokey public-key cryptography, as described in the Authentication Options page. If any bits are set and the association has previously been marked reachable, the packet is discarded; otherwise, the originate and receive timestamps are saved, as required by the NTP protocol, and processing continues.

TEST10 through TEST 13 checks the health of the association itself. If any bits are set, the packet is discarded; otherwise, the saved variables are passed to the clock filter and mitigation algorithms.

The flash bits for each test are defined as follows.

0x0001 TEST1: *Duplicate packet.* The packet is at best a casual retransmission and at worst a malicious replay.

0x0002 TEST2: *Bogus packet.* The packet is not a reply to a message previously sent. This can happen when the NTP daemon is restarted and before somebody else notices.

0x0004 TEST3: *Protocol Unsynchronized.* One or more timestamp fields are invalid. This normally happens when the first packet from a peer is received.

0x0008 TEST4: *Access is denied.* See the Access Control Options page in the reference implementation documentation.

0x0010 TEST5: *Cryptographic authentication fails.* See the Authentication Options page in the reference implementation documentation.

0x0020 TEST6: *The server is unsynchronized.* Wind up its clock first.

0x0040 TEST7: *Bad Synchronization or Stratum.* The server stratum is at the maximum (greater than 15.) It is probably unsynchronized and its clock needs to be wound up.

0x0080 TEST8: *Autokey Error.* The autokey protocol has detected an authentication failure. See the autokey documentation included in the reference implementation.

0x0100 TEST9: *Cryptographic Error.* This is likely to be a problem with the keys.

0x0200 TEST10: *Peer is unsynchronized or at an incorrect stratum.*

0x0400 TEST11: *Peer synchronization distance exceeded.* The synchronization distance from a host to a primary/root server through the specified peer has exceeded a threshold.

0x0800 TEST12: *Peer synchronization loop detected.* A synchronization loop has been detected, that is, the peer being used as a host's source is synchronization is itself deriving its time from the host being synchronized. Change the sources of synchronization in the configuration file.

0x1000 TEST13: *Peer unreachable.* The peer is not replying to host's requests for time. This usually happens when the reachability register becomes zero, that is, the peer has not replied to the last eight messages sent by the host.

C Reference Implementation Configuration

Following is a description of the configuration commands in the NTPv4 reference implementation. There are two classes of commands, configuration commands that configure an association with a remote server, peer or reference clock, and auxiliary commands that specify environmental variables that control various related operations.

C.1 Configuration Commands

The various modes are determined by the command keyword and the required IP address. Addresses are classed by type as (s) a remote server or peer (IPv4 class A, B and C), (b) the broadcast address of a local interface, (m) a multicast address (IPv4 class D), or (r) a reference clock address (127.127.x.x). The options that can be used with these commands are listed below.

If the Basic Socket Interface Extensions for IPv6 (RFC-2553) is detected, support for the IPv6 address family is generated in addition to the default support of the IPv4 address family. IPv6 addresses can be identified by the presence of colons ":" in the address field. IPv6 addresses can be used almost everywhere where IPv4 addresses can be used, with the exception of reference clock addresses, which are always IPv4. Note that in contexts where a host name is expected, a -4 qualifier preceding the host name forces DNS resolution to the IPv4 namespace, while a -6 qualifier forces DNS resolution to the IPv6 namespace.

There are three types of associations: persistent, preemptable and ephemeral. Persistent associations are mobilized by a configuration command and never demobilized. Preemptable associations, which are new to NTPv4, are mobilized by a configuration command which includes the prempt flag and are demobilized by timeout or error. Ephemeral associations are mobilized upon arrival of designated messages and demobilized by timeout or error.

server ***address [options ...]***
peer ***address [options ...]***
broadcast ***address [options ...]***

manycastclient ***address [options ...]***

These four commands specify the time server name or address to be used and the mode in which to operate. The address can be either a DNS name or a IP address in dotted-quad notation. Additional information on association behavior can be found on the Association Management page in the reference implementation documentation.

server
For type s and r addresses (only), this command normally mobilizes a persistent client mode association with the specified remote server or local reference clock. If the preempt flag is specified, a preemptable association is mobilized instead. In client mode the client clock can synchronize to the remote server or local reference clock, but the remote server can never be synchronized to the client clock. This command should NOT be used for type b or m addresses.

peer
For type s addresses (only), this command mobilizes a persistent symmetric-active mode association with the specified remote peer. In this mode the local clock can be synchronized to the remote peer or the remote peer can be synchronized to the local clock. This is useful in a network of servers where, depending on various failure scenarios, either the local or remote peer may be the better source of time. This command should NOT be used for type b, m or r addresses.

broadcast
For type b and m addresses (only), this command mobilizes a persistent broadcast mode association. Multiple commands can be used to specify multiple local broadcast interfaces (subnets) and/or multiple multicast groups. Note that local broadcast messages go only to the interface associated with the subnet specified, but multicast messages go to all interfaces.

In broadcast mode the local server sends periodic broadcast messages to a client population at the address specified, which is usually the broadcast address on (one of) the local network(s) or a multicast address assigned to NTP. The IANA has assigned the multicast group address IPv4 224.0.1.1 and IPv6 ff05::101 (site local) exclusively to NTP, but other nonconflicting addresses can be used to contain the messages within administrative boundaries. Ordinarily, this specification applies only to the local server operating as a sender; for operation as a broadcast client, see the broadcastclient or multicastclient commands below.

manycastclient
For type m addresses (only), this command mobilizes a preemptable manycast client mode association for the multicast group address specified. In this mode a specific address must be supplied which matches the address used on the manycastserver command for the designated manycast servers. The NTP multicast address 224.0.1.1 assigned by the IANA should NOT

be used, unless specific means are taken to avoid spraying large areas of the Internet with these messages and causing a possibly massive implosion of replies at the sender.

The manycastclient command specifies that the host is to operate in client mode with the remote servers that are discovered as the result of broadcast/multicast messages. The client broadcasts a request message to the group address associated with the specified address and specifically enabled servers respond to these messages. The client selects the servers providing the best time and continues as with the server command. The remaining servers are discarded as if never heard.

C.1.1 Command Options

autokey

All packets sent to and received from the server or peer are to include authentication fields encrypted using the autokey scheme described in the Authentication Options page. This option is valid with all commands.

burst

When the server is reachable, send a burst of eight packets instead of the usual one. The packet spacing is normally 2 s; however, the spacing between the first and second packets can be changed with the calldelay command to allow additional time for a modem or ISDN call to complete. This option is valid with only the server command and is a recommended option with this command when the maxpoll option is 11 or greater.

iburst

When the server is unreachable, send a burst of eight packets instead of the usual one. The packet spacing is normally 2 s; however, the spacing between the first and second packets can be changed with the calldelay command to allow additional time for a modem or ISDN call to complete. This option is valid with only the server command and is a recommended option with this command.

key ***key***

All packets sent to and received from the server or peer are to include authentication fields encrypted using the specified key identifier with values from 1 to 65534, inclusive. The default is to include no encryption field. This option is valid with all commands.

minpoll ***minpoll***

maxpoll ***maxpoll***

These options specify the minimum and maximum poll intervals for NTP messages, in seconds as a power of two. The maximum poll interval defaults to 10 (1,024 s), but can be increased by the maxpoll option to an upper limit of 17 (36.4 h). The minimum poll interval defaults to 6 (64 s), but can be decreased by the minpoll option to a lower limit of 4 (16 s). These option are valid only with the server and peer commands.

noselect
Marks the server as unused, except for display purposes. The server is discarded by the selection algorithm. This option is valid only with the server and peer commands.

preempt
Specifies the association as preemptable rather than the default persistent. This option is valied only with the server command.

prefer
Marks the server as preferred. All other things being equal, this host will be chosen for synchronization among a set of correctly operating hosts. See the Mitigation Rules and the prefer Keyword page for further information. This option is valid only with the server and peer commands.

true
Force the association to assume truechimer status; that is, always survive the selection and clustering algorithms. This option can be used with any association, but is most useful for reference clocks with large jitter on the serial port and precision pulse-per-second (PPS) signals. Caution: this option defeats the algorithms designed to cast out falsetickers and can allow these sources to set the system clock. This option is valid only with the server and peer commands.

ttl *ttl*
This option is used only with broadcast server and manycast client modes. It specifies the time-to-live ttl to use on broadcast server and multicast server and the maximum ttl for the expanding ring search with manycast client packets. Selection of the proper value, which defaults to 127, is something of a black art and should be coordinated with the network administrator.

version *version*
Specifies the version number to be used for outgoing NTP packets. Versions 1-4 are the choices, with version 4 the default. This option is valid only with the server, peer and broadcast commands.

C.2 Auxiliary Commands

broadcastclient *[novolley]*
This command enables reception of broadcast server messages to any local interface (type b) address. Ordinarily, upon receiving a message for the first time, the broadcast client measures the nominal server propagation delay using a brief client/server exchange with the server, after which it continues in listen-only mode. If the novolley keyword is present, the exchange is not used and the value specified in the broadcastdelay command is used or, if the broadcastdelay command is not used, the default 4.0 ms. Note that, in order to avoid accidental or malicious disruption in this mode, both the server and client should operate using symmetric key or public key authentication as described in the Authentication Options page. Note that the novolley keyword is incompatible with public key authentication.

manycastserver ***address [...]***

This command enables reception of manycast client messages to the multicast group address(es) (type m) specified. At least one address is required. The NTP multicast address 224.0.1.1 assigned by the IANA should NOT be used, unless specific means are taken to limit the span of the reply and avoid a possibly massive implosion at the original sender. Note that, in order to avoid accidental or malicious disruption in this mode, both the server and client should operate using symmetric key or public key authentication as described in the Authentication Options page.

multicastclient ***address [...]***

This command enables reception of multicast server messages to the multicast group address(es) (type m) specified. Upon receiving a message for the first time, the multicast client measures the nominal server propagation delay using a brief client/server exchange with the server, then enters the broadcast client mode, in which it synchronizes to succeeding multicast messages. Note that, in order to avoid accidental or malicious disruption in this mode, both the server and client should operate using symmetric key or public key authentication as described in the Authentication Options page.