Scalable, High Speed, Internet Time Synchronization

Advanced Research Projects Agency
Contract DABT 63-95-C-0046

Quarterly Progress Report
1 September 1995 - 30 November 1995

David L. Mills
Electrical Engineering Department
University of Delaware

1. Introduction

The work reported during this interval consists of a status report on the development of a generic autonomous configuration facility suitable for distributed, ubiquitous protocols such as the Network Time Protocol (NTP) and others of that nature. Also included are formal specification to support new protocol modes as described in the contract proposal. The primary function of these revisions is to support a new mode of operation called NTP Distributed Mode. It is designed for use in dense forests of time-synchronized peers were the highest degree of redundancy is required, while minimizing the number of synchronizing messages required. With N peers in the NTP subnet, the number of messages scales as N2 for unicast modes, but reduces to N for multicast modes, including the new mode.

One of the motivations for the development of distributed mode is as a vehicle for investigation of NTP related artifacts, such as asymmetric routes, security schemes and Byzantine faults. This note speaks only to the design and implementation of the distributed mode itself; later notes will speak to the applications that this mode is designed to support.

A set of technical memoranda has been produced describing technology suitable for calibrating the performance of computers and packet-switching networks. These memoranda have been installed in the web page collection for Prof. David L. Mills. (http://www.udel.edu/~mills). These are

Mills, D.L. Time and Time Interval Measurement with Application to Computer and Network Performance Evaluation. Electrical Engineering Technical Memorandum, January 1996, 17 pp. ftp://louie.udel.edu/pub/people/mills/memos/memo96a.ps

> This memorandum surveys and analyzes new and existing paradigms for synchronizing test instruments deployed in a distributed configuration for measuring performance of telecommunications systems, such as ATM switches, SDH transmission equipment and ancillary supporting devices. These instruments often take the form of a general purpose PC or workstation equipped with special purpose interfaces suitable for nonintrusive monitoring of telecommunication equipment operations. However, the same methodology developed for these systems should be usable in more general applications where time and/or frequency synchronization is necessary between components of a distributed system.

Mills, D.L. A Kernel Model for Precision Timekeeping. Electrical Engineering Technical Memorandum, January 1996, 28 pp. ftp://louie.udel.edu/pub/people/mills/memos/memo96b.ps

This memorandum is an update of a previous memorandum and RFC on precision time modifications the Unix kernels. The update includes material learned from integrating these modifications in DEC and Sun kernels for symmetric multiprocessor systems.

Mills, D.L. A Kernel Programming Interface for Precision Time Signals. Electrical Engineering Technical Memorandum, January 1996, 3 pp. ftp://louie.udel.edu/pub/people/mills/memos/memo96c.ps

This memorandum proposes a programming interface for the generic Unix kernel which provides support for external timing signals, such as the pulse-per-second signal generated by some radio clocks and cesium oscillators.

2. Autonomous Configuration

This section describes the basic problems in automatically configuring a very large network of clients and servers. Given such a network interconnected by communication links, a typical network design problem is to construct minimum spanning trees (MST) or shortest-path spanning trees (SPT) rooted at designated vertices in the network. Such problems are often found in the design of network routing algorithms and efficient algorithms have been developed to deal with them. Another common network design problem involves the topological configuration of terminal concentrator networks, i.e., where to put the concentrators in a large network of terminals and application processors in order to minimize communication costs.

Other examples include designing networks of minimum cost or capacity while meeting specific performance requirements, such as delay. In most cases, the design process involves determining the topology of the communication subnet, i.e., the location of the nodes, the topology of the links and the capacity of each link. The cost function typically involves a) minimizing the average delay for the dissemination of data in the network, b) ensuring the integrity of network service in the event of node and link failures, or c) minimizing the total capital investment and operating costs while meeting criteria a) and b) above.

In general, many real-life situations might end up being very broad problems that may be quite difficult to formulate precisely. However, in many cases, the situation can be considerably simplified. For example, the problem may be simplified through the use of hierarchical principles. More frequently, there may be a natural decomposition of the problem - for example, a subnet design problem may be naturally decoupled from the overall network design problem. This can be found in cases where the network is already in place and a subnet is to be designed to optimize certain parameters. It is seldom the case that one ends up with a clean problem that can be solved exactly in a reasonable time and one typically looks for approximation algorithms through heuristic methods that combine theory, trial and error and basic common sense.

2.1 The Subnet Design Problem

In this problem, we are given a network of nodes along with the distance (cost) of each link between two nodes. A relatively small subset of nodes are designated primary servers and the remaining nodes are either secondary servers or clients. Servers are responsible for passing information such as clock offsets, etc., to other servers and clients. Primary servers obtain information from an external source, whereas secondary servers obtain information from other servers in the network. We would like to design the topology of the subnet such that every node in the network has access to information from at least m primary severs while minimizing the total cost in the network. The cost of a path from a client to its designated server is the cost assigned that client. The total cost is the sum of all path costs. The problem corresponds to building SPTs rooted at each of the m primary servers, where the cost of any path may be constrained to be not greater than p and the degree of paths incident at any node may be constrained to be not greater than k.

The rationale behind having every node access to m primary servers is to ensure adequate redundancy, so that the node can tolerate up to $m - 1$ server failures and still be able to access the information. Ensuring that the cost between each primary server and any of its clients is within p guarantees that no node will be forced to access information at a cost greater than p. If, for example, the cost function were to represent delay, then this would guarantee that all nodes would receive information within the specified delay interval. The degree constraint k ensures that no node has greater than $k - 1$ children, which may be necessary to manage the processing load on each server.

Examples of instances where such constraints may be observed in real-life are as follows:

1. Given a network of World Wide Web servers and clients, along with a set of strategically located caching sites, whose task is to avoid repetitive transfer of real-time data over international links, the objective would be to design a suitable distribution mechanism by which the cache servers and clients organize themselves into a hierarchy to minimize the quantity of information transfered. An added constraint could be that no server in the hierarchy is responsible for distributing information to more than k clients and never responds in more than p seconds..

2. Given a set of NTP clients and severs, a certain subset of which may be designated as primary NTP servers, the objective is build a self-organizing, hierarchical time distribution mechanism such that the sum of the synchronization distances of all nodes in the network and the primary server they receive time from is minimized. The constraints in this case are that no node may be at a distance greater than p seconds from its primary server and no node may be responsible for distributing information to more than k children. In addition, each node must receive time from at least m other primary servers in order to maintain redundancy and achieve specified tolerance to faults at all times.

Given an outline of the problem, an effective model for study can be developed using graph theory. Suppose we are given a directed graph $G = (V, E)$, where V represents the set of vertices or nodes, and $E = (i, j)$, where i, j are nodes, is the set of edges or links connecting them. Each directed edge in G is assigned a distance or cost $C(i, j)$ of communicating between them. We assume that the graph is connected, i.e., there exists a set of edges between any two nodes, and complete, i.e., there is a path with nonzero cost from each node of a designated subset of nodes, called the root set, to every other node in G.

The objective is to construct optimum trees or subgraphs subject to the constraints given above. Formulating this problem as an integer programming problem is not trivial, but can be achieved along the following lines:

Minimize $\sum P(i,j)Y(i,j)$, over all nodes *i* and root nodes *j*. where $P(i,j)$ is the sum of all edge costs between a non-root node *i* and root node *j, i* i.e.,

$$P(l,j) = \sum_{l}[P(l,j) + C(l,j)X(l,j)] \text{, subject to: } \sum_{l} X(j,l) \geq 1 \tag{1}$$

$$\sum X(i,l) + \sum_{l} X(l,i) \leq k \tag{2}$$

$$\sum_{i} Y(i,j) > m - 1 \tag{3}$$

$$P(i,j) < p \tag{4}$$

$$Y(i,j) = \{0, 1\} \tag{5}$$

$$X(i,l) = \{0, 1\} \tag{6}$$

A brief explanation of the constraints will help in understanding the problem. Constraint (1) ensures that every node has at least one incoming edge, i.e., it is part of at least one tree. This constraint may be modified to make it $E = m$, in which case the there would be a forest of *m* disjoint trees that a single node would be part of. This would be a much harder constraint to meet, especially in sparse graphs, and hence it is easier to solve for the case where there is at least one edge incident into a node. (2) ensures that a single node cannot have a degree greater than *k*, while (3) ensures that every node has access to at least *m* primaries. Constraint (4) is a restriction on the sum of the weights between each root node and its children. (5) and (6) represent which root nodes are associated with each node, and which edges are chosen in the graph to form the various trees.

2.2 Complexity Issues

It is trivial to see that this problem is a member of the NP-Hard class of problems. A comprehensive theoretical analysis of the problem would provide a good insight into its complexity implications. Recently, there has been considerable interest in multi-objective optimization problems and a significant effort is underway to devise good approximation algorithms for this class of problems. It is prudent at this point to take a step back and see if it is possible to break the problem into constituent subproblems. It may be that each one of those may be NP-Hard in their own accord, but it is a useful exercise to be able to break a complex problem into smaller components and analyze each one individually in order to gain some insight into the nature of the problem.

If the bound on the degree of each node is removed, along with the bound on the maximum distance of each node to a root node, then the problem reduces to assigning each node to m root nodes. This is a trivial problem to solve and the optimal solution is obtained by simply ranking all the root nodes of every node according to nondeceasing cost and choosing the m root nodes that have the lowest cost. Adding any of the other constraints makes it much harder to solve.

A couple of variations of the above problem have been studied in the literature. The k-center problem involves fixing the number of root nodes at k, and trying to minimize the maximum distance from any node to its associated root node. It has been shown that there exists an approximation algorithm with a performance guarantee of 2 for the k-center problem. The p-dominating set problem is a dual of the k-center problem, where the objective is to minimize the number of root nodes, given that no node in the graph can be at a distance greater than p from its associated root node. This problem is harder to solve and the best known approximation algorithm for this class of problems has a performance guarantee of $O(\log N)$.

Let us consider the case where there is only a single root node. One subproblem would be to construct minimum cost trees rooted at this node such that each other node in the tree has degree no greater than *k*. No constant-factor approximation algorithms have yet been found for this problem; however, there is an approximation algorithm for this problem. Another similar problem would be to construct a *k*-degree tree from the root nodes such that the maximum cost of the tree is minimized. Again, no constant-factor $O(\log N)$ approximation algorithm has been found for this problem.

We are currently studying the k-degree minimum cost problem and attempting to come up with a constant-factor approximation algorithm. One technique that seems to hold good promise here is a primal-dual approach. This mechanism involves specifying the problem as an integer programming problem and then formulating the dual of the problem. The algorithm then works by successively satisfying constraints in the primal and dual problems, while approaching the optimal solution. These algorithms have been shown to run in polynomial time, thus making it feasible for use in very large networks.

3. NTP Distributed Mode

The following text is designed to be merged at some later time in the NTP Version 3 specification document RFC1305 as part of the evolution to Version 4 of the protocol. These changes are in addition to those specified in the previous quarterly report. Additional changes are expected in order to define the packet formats and processing details unique to this mode.

It is the intent that the above processing rules be added to the existing rules specified in RFC1305, as amended by those in the previous quarterly report. These rules will in principle result in no changes in the operation of existing NTP servers and clients and should in general be transparent to all prior versions of the protocol. The new rules do specify detailed behavior in configurations that formerly were considered erroneous, but generally resulted in correct clock synchronization. Of primary utility is the specification of the distributed mode operations, which have no prior definition or usage history. However, the above rules do not specify precisely how such implementations should be built and how they should respond. This is left for future work.

NTP Distributed Mode is designed to capture timestamps from a subnet of time servers, or peers, and construct a database from which individual offset and delay measurements can be computed between any pair of peers in the subnet. In this mode, each peer transmits a multicast message containing the time the message is sent, together with state variables determined directly and indirectly from multicast messages received from other subnet peers.

The term direct timestamps refers to measurements made by a peer relative to its own local clock; that is, any measurement error must be due to either errors in setting the local clock or operating system jitter accumulated by the local clock reading routines. The term “indirect timestamps” refers to measurements made by one peer and subsequently distributed to another peer using the protocol model described below. Indirect timestamps are bundled with identification information in the form of tuples and then encapsulated in multicast messages sent to all members of the subnet. In order to calculate the error budget accurately with indirect timestamps, it is necessary to know additional details, such as the precision of the local clock, the synchronization distance, etc. This information is learned from multicast messages sent by each peer in the subnet.

NTP distributed mode uses direct and indirect information provided by each peer in the subnet, but a particular NTP message may not contain data for all other peers. The transmit timestamp in the NTP header provides the time of transmission relative to the sending peer clock, while the time of reception at every other peer is recorded as the arrival timestamp for purposes of NTP calculations. When a peer receives an NTP multicast message, it searches the list of timestamp tuples in order to find the one associated with its own previously transmitted message. Since it may not be possible to include all timestamp tuples in every message transmitted, peers may have to wait up to several messages in order to find its own timestamps and complete its clock offset and roundtrip delay calculations. Therefore, the frequency of multicast messages may be higher than in subnets without this capability, but the frequency will generally be much lower than if distributed mode was not used.

It is important to realize that offset/delay measurements made between the same pair of peers in the NTP unicast modes and NTP multicast modes can differ, since the unicast and broadcast spanning trees may differ due to shared-tree topologies, etc. For the highest accuracies, NTP unicast modes (client-server or symmetric) must be used. In these modes each peer uses its direct timestamps together with its neighbor’s indirect timestamps to accurately calibrate the clock offset and roundtrip delay. However, provisions are made in the protocol model to measure the offset discrepancy between the unicast and multicast modes; however, such measurements are made relatively infrequently and may not reflect the state of the network at any particular time. It may in fact be most desirable to run unicast and multicast protocols at the same time, using unicast protocols to fine-tune the local clock time and frequency and use multicast modes in order to detect and correct for asymmetrical paths, etc.

This note explores design issues involved in the NTP distributed mode and discusses details of protocol and packet format.The following considerations are relevant:

1. The message formats should be as close to existing formats as possible without compromising the accuracy or precision of the representation.

2. Each timestamp tuple should be self contained and not depend on other information in the same or different messages.

3. Depending on accuracy expectations, it may be necessary to run unicast and distributed modes simultaneously. In such cases, the unicast timestamps are used to directly synchronize peer clocks, with distributed mode used to verify consistency, resolve asymmetrical delays and so forth.

4. Messages used to exchange distributed timestamps may contain only a fraction of the available timestamp tuples. Successive messages may include subsets of the tuples depending on maximum message size, rate of change of information, and maybe other factors.

5. The precision of calculation for both clock offset and roundtrip delay using distributed mode should be comparable to the precision using unicast or multicast modes.

While the specific scheme used to authenticate NTP packets may be changed in future to align with generic IP schemes, the packet formats should not prohibit use of the current NTP authentication scheme.

4. NTP Timestamps and Data Base

In order to calculate clock offset and roundtrip delay between two peers, the NTP specification calls for four timestamps, two generated by each peer between which these measurements are to be made. Each peer can calculate the offset and delay relative to the other by recording two timestamps and receiving the other two from the other peer in an NTP message. Subnet peers other than those directly involved in such measurements can also perform these calculations if the timestamps are made available in multicast messages from each peer and with certain additional information as described below.

Presume peers A and B are exchanging multicast messages in order to set their clocks and that peer X overhears these messages. Peer A needs four timestamps in order to calculate the clock offset and roundtrip delay relative to B:

T1 originate timestamp (request sent at A)
T2 receive timestamp (request received at B)
T3 transmit timestamp (response sent at B)
T4 destination timestamp (response received at A)

Peer A maintains three state variables for every other peer, including B:

S1 latest originate timestamp
S2 latest transmit timestamp
S3 latest destination timestamp

In peer A, S1 remembers the time according to the A clock when the most recent request was sent; i.e., the originate timestamp, T1. This variable is used only in unicast modes to verify the response matches the request. S2 remembers the time according to the B clock when the most recent reply was sent; i.e., the transmit timestamp included in that reply, T3. S3 remembers the time according to the A clock when the reply was received; i.e., the destination timestamp, T4.

In unicast modes, peer A sends a request to peer B piggybacked with a response from a prior request from B. Peer B does the same thing in a symmetric manner. The packet variables are:

P1 originate timestamp (request sent at A)
P2 transmit timestamp (prior response/request sent at B)
P3 receive timestamp (prior response/request received at A)

The NTP unicast message sent by A contains the originate timestamp P1 (S1) of the current request, plus the transmit timestamp P2 (S2) and destination timestamp P3 (S3) of the most recent response received. Upon arrival at B, the P1 field in the message becomes the T3 value for B, the P2 field becomes the T1 value and the P3 field becomes the T2 value. The T4 value for B is captured from the local clock at B upon arrival of the message. Following calculation of the clock offset and roundtrip delay, the T3 timestamp replaces state variable S2 at B and the T4 timestamp replaces state variable S3. When B sends a response/request message to A, the same things happen with A and B in the above description interchanged.

Each time a new round of timestamps $T1 - T4$ is updated, the roundtrip delay and clock offset is calculated:

clock offset $\frac{(T2 - T1) + (T3 - T4)}{2}$, roundtrip delay $(T4 - T1) - (T3 - T2)$.

Note that, due to the symmetry of the exchange, both A and B can independently calculate clock offset and roundtrip delay of their local clock with respect to their peer clock. In practice, each peer independently sends messages with current state variables to the other peer at designated poll intervals. It doesn't matter if messages are lost, since the latest state variables and timestamps are used for each one. It doesn't matter if messages are duplicated, since the receiver can detect this and toss the duplicates out. Finally, note that X, overhearing a single message from A and a single message from B can also calculate the clock offset and roundtrip delay of either peer relative to the other.

4.1 Sending Distributed Mode Timestamps

In multicast modes, separate state variables S2 and S3 are maintained for every peer in the subnet. Periodically, these variables are collected and transmitted by peer X in a multicast message, which is presumably heard by all members of the subnet. These variables are transmitted as a 20-octet tuple consisting of the IP group address, followed by the two state variables S2 and S3. The IP address is the address of the peer Y which sent a message at some prior time S2 and was received by the peer X at time S3. Ordinarily, the IP address is the address of the peer Y sending the data from which the timestamps are derived, not necessarily the one X sending the tuple. The values of the state variables S2 and S3 are derived from the most recent request/reply received from the peer in the manner described for unicast modes.

There is no need to hold internal state other than S2 and S3 for each peer in the subnet; the S1 variable is not used in multicast modes. Therefore, some means of authentication is necessary in order to prevent spoofing attacks. Experience has shown this to be necessary in any multicast mode, including NTP distributed mode. Note that, while S2 and S3 are transmitted for each peer separately, the transmit timestamp of the multicast message itself is included in the NTP header, which is transmitted in both unicast and multicast modes.

It is the intent in the protocol design that each timestamp message contain some number, but perhaps not all, tuples for the entire subnet. The maximum number of tuples that can be included in a message depends on the MTU for the particular connected network. Determining the MTU is outside the scope of this memo; however, a working maximum should be in the neighborhood of 20 for a 512-octet IP datagram. In any case, the length of the NTP message itself can affect the accuracy of synchronization, since the time to transmit the message itself is included in the roundtrip delay. Therefore, whatever procedures are used to pack the messages should insure that the messages are all the same length.

4.2 Receiving Distributed Mode Timestamps

In order to independently calculate clock offset and roundtrip delay between A and B, peer X must be able to construct consistent timestamps T1, T2, T3 and T4 for either or both A and B. In general,~if calculations for A with respect to B are needed, a multicast message from B reveals one pair of timestamps (T1 and T2), while one from A reveals the other pair (T3 and T4). In order to do this, it must insure that the data collected at X using a single message from A and a single message from B are consistent with a single exchange of messages between A and B.

Let (T1, T2) and (T3, T4) be the two pairs of timestamps, the first from peer B, the second from peer A. There are two major cases distinguished by whether the messages cross in transit across the network. If they do not cross and the A and B clocks have not been reset and the frequency error is within bounds yet to be determined, then by construction $T3 \geq T2$ and $T4 \geq T1$. In this case, there are two subcases:

1. $(T3 - T2) < (T4 - T1)$. The request was sent by A, the reply by B. In this case, the offset/ delay data are calculated relative to peer A.

2. $(T4 - T1) < (T3 - T2)$. The request was sent by B, the reply by A. In this case, the offset/ delay data are calculated relative to peer B.

If the messages do cross, then by construction $T3 < T2$ and $T4 < T1$. This case is normally infrequent, but could occur if the network delays are significant with respect to the poll intervals. While it might appear that this case could lead to misleading results, examination of the above equations for clock offset and roundtrip delay show that correct results are obtained, as long as the signed intermediate values in the calculations are handled correctly. Since in the 64-bit NTP timestamp format the high order sign bits are set, this requires some care.

4.3 Implementation Details

Some implementation details follow. First, a database is needed to hold data used by the offset/ delay calculations. The database is indexed by each pair of subnet peers, most conveniently in the form of a matrix *i, j*, where *i, j* range over the peers in the subnet. The *i*th row is sent by the *i*th peer, possibly in several pieces. The *j*th element of that row is accumulated from messages received from the *j*th peer.

Each element of the matrix is a data structure similar to the existing peer data structure; that is, it contains most of the variables and data structures used in the unicast modes, including the clock filter, offset and delay estimates, etc., as the original peer data structure. However, the new structure is not used to send packets, although it should include reachability information. Note that each peer pair is represented twice in the matrix, once for each direction of propagation. It does not seem useful to include peer data structures *i, i* along the main diagonal.

Some of the variables necessary to populate the structure for the *j*th element in each of the rows is extracted from NTP messages sent by the *j*th host itself, including the precision, root delay, and so forth. In order to minimize the total size of the structure, it may be necessary to maintain a single copy of these variables together with a system of pointers. In this scheme, a structure would be built for each peer holding common variables and the row of timestamps *i, j* included as a vector in that structure. Note however, that the per-peer variables for the clock filter, etc., must be maintained in the vector.

It seems most useful to transmit the tuples described previously in NTP multicast messages with mode field specifying IP multicast mode. For older version servers which ignore the tuples included in the message following the conventional NTP header, these would look just like ordinary multicast mode messages, which could be processed or ignored as indicated. An old-style server could go through the initial calibration phase of the multicast client mode as described in the last report, since the state machine supports all old modes.

NTP multicast-mode messages with tuples following the NTP header are processed in a special way. Assume the sending peer is i and receiving peer is *j*. First, the transmit timestamp P3 in the NTP message header together with the destination timestamp are used to update the S2 and S3 state variables associated with the *i, j* structure in the matrix. The other header fields are ignored for this step. Proceeding in this way, the entire column *i, j* of direct timestamps for all sender peers *i* will be filled in by messages these peers as they arrive. Each time a structure is updated, it is marked for later processing.

Next, the indirect timestamp tuples are processed. For each one, let k be the origin as recorded by the ith peer sending the message and received by the jth peer as before. The tuple is then used to update the S2 and S3 state variables associated with the *i, j* structure and that structure marked for later processing. Since a peer doesn't send indirect timestamps for itself, the entry *i, i* never occurs for either direct or indirect timestamps. Proceeding in this way, the entire column *i, j* is filled in for all sender peers *k* as determined by peer *i* and later reported to peer *j*.

Ad predetermined intervals, each peer *i* sends the entire ith column of direct timestamps encapsulated in NTP multicast messages as described previously. Except for small differences due to the order of message arrival, the matrices in all subnet peers should eventually converge to the same values. The interesting data are of course these presumed small differences, which can be used to improve accuracy and reliability in ways to be explored.

The remaining processing does not have to be done at the time of arrival of each message, but could be delayed for offline processing at slack moments. For each peer *i, j* whose state variables have changed, it is necessary to go through the suite of NTP algorithms, including the clock filter, intersection, clustering and combining algorithms, separately in order to calculate the clock offset, roundtrip delay and dispersion. A convenient way to do this may be to substitute each tuple of IP address and state variables for the data that would be included in a NTP message between each pair of peers and simulate the calculations.

Consider two peers i and j and the set of state variables for *i, j* and *j, i* . From these, the source and destination IP addresses are obtained from *i, j* and *j, i* respectively, while one pair of timestamps T1 and T2 are obtained from *j, i* and the other pair T3 and T4 are obtained from *i, j*. Processing continues as in the unicast mode case. Note that, in the case of the ith row, the computations for *i, j* and *j, i* represent paths between the ith peer and each of the *j* peers from which multicast messages have been received, so these are direct measurements.

Note that the above scheme does not preserve the originate timestamp check used in the unicast modes to authenticate the reply. While it would in principle be possible to preserve this datum and include it in the tuple data, this would considerably increase the storage and transmission overheads. Also in principle, the authentication model can be based on other cryptographic data, such as public and private keys, that would seem naturally appropriate for proliferated multicast protocols such as this.

5. Association Management

In principal, there is no configuration management required in NTP distributed mode. Upon activation, an NTP peer simply starts broadcasting its current (possibly empty) vector of state variables. As more peers are activated, each hears the others and builds a dynamic matrix {i,~j} as described above. Each new peer not heard before causes a new row and column to be created in the matrix. If a peer hasn't been heard from for awhile, as determined by the usual reachability algorithm, its row and column are expunged from the matrix. Probably, the best approach for the reachability algorithm is to use only the direct information, avoiding the information implied in the indirect timestamps.

There are some interesting issues to be resolved on how the NTP distributed mode interacts, if at all, with the older unicast and multicast modes. There is some concern on the accuracy of the distributed mode, since it relies on the embedded multicast spanning tree (possibly shared) and the pragmatic observation that this tree is very likely not directly derived from the underlying a unicast spanning tree due to administrative meddling. Accordingly, means should be explored similar to the older multicast/unicast scheme, in which the protocol executes a preliminary (and possibly repeated at long intervals) calibration procedure before assuming a listen-only mode. In the distributed mode, peers both send and receive; so, in principle the only detractions to best attainable accuracy are routing artifacts created by the multicast spanning tree. The success of this scheme depends on the jitter and routing churn of the unicast and multicast trees; indeed, one of the applications the distributed mode enables is the calibration and assessment of routing algorithms that produce these trees.

6. Packet Formats and Parsing

Inclusion of indirect timestamp tuples following the NTP header does not create format problems, since the NTP header is fixed length; however, the scheme could conflict with the authentication scheme long used with NTP. In this scheme, the message authentication code (MAC) is inserted in the message following all other data in the message and the presence of the MAC is used to signal that the message is authenticated. No other indicator of authentication is defined in the header. When used in the DES-CBC mode, the MAC is 96 bits, so cannot be confused with a distributed mode tuple. However, when used in the MD5 mode, the MAC is 160 bits, which is unfortunately also the length of a tuple. This creates an ambiguity in the specification, unless a count field is included immediately following the fixed-length header and before the first tuple. This seems to be the course of least resistance.

7. Plans for Next Quarter

During the next quarter, we expect to begin revision of the NTP protocol specification to conform to the above rules. We also expect to begin work on necessary security features to support the new mode and simplify the keying operations required in a widely deployed system. This will be reported in a subsequent quarterly report. Finally, we expect to begin work on algorithms to automatically configure a subnet based on defined metrics and constraints as described in the contract proposal.