RC24361 (W0709-128) September 28, 2007
Computer Science

IBM Research Report

Almost Peer-to-Peer Clock Synchronization

Ahmed Sobeih
University of Illinois at Urbana-Champaign
Department of Computer Science
Urbana, IL 61801 USA

Michel Hack, Zhen Liu, Li Zhang
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598 USA

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

Almost Peer-to-Peer Clock Synchronization

Ahmed Sobeih[1], Michel Hack[2], Zhen Liu[2], and Li Zhang[2]

[1]University of Illinois at Urbana-Champaign
Department of Computer Science
Urbana, IL 61801 USA
sobeih@uiuc.edu

[2]IBM T.J. Watson Research Center
19 Skyline Drive
Hawthorne, NY 10532 USA
{hack,zhenl,zhangli}@us.ibm.com

"All animals are equal, but some animals are more equal than others." George Orwell (1945)

Abstract

In this paper, an almost peer-to-peer (AP2P) clock synchronization protocol is proposed. AP2P is almost peer-to-peer in the sense that it provides the desirable features of a purely hierarchical (client/server) clock synchronization protocol while avoiding the undesirable consequences of a purely peer-to-peer one. In AP2P, a unique node is elected as a leader in a distributed manner. Each non-leader node adjusts its clock rate based on message exchanges with its neighbors, taking into consideration that neighbors that are closer to the leader have more effect on the adjustment than the neighbors that are further away from the leader. We compare the performance of AP2P with that of the Server Time Protocol (STP), which is a purely hierarchical clock synchronization protocol. Simulation results, which have been conducted on several network topologies, have shown that AP2P can provide a clock synchronization accuracy that is indistinguishable from that of STP. Furthermore, AP2P is more fault-tolerant because it can recover from certain types of failures that STP cannot recover from.

1. Introduction

Clock synchronization is important for a wide variety of applications, e.g. banking transactions, log management, bandwidth usage and network fault detection. For instance, Cisco routers use the Network Time Protocol (NTP) [15] to compare time logs, which is essential for tracking security incidents, analyzing faults and troubleshooting [1]. In multi-hop wireless ad hoc networks, clock synchronization is necessary for several operations, such as power management and frequency hopping in the IEEE 802.11 standard [3]. In wireless sensor networks, information dissemination paradigms (e.g., [11]) require time synchronization.

In this paper, clock synchronization refers to the mechanisms and protocols used to maintain mutually consistent time-of-day clocks in a coordinated network of computers. The intent is to provide the illusion of a global time-of-day clock that is strictly monotonic as observed by any node in the network: if, at time T_1, node A asks node B to report its current time T_2, and the reply is received at node A at time T_3, then we would like to guarantee that $T_1 < T_2 < T_3$. (For this concept to make physical sense, we shall rule out interplanetary networks with nodes moving about at relativistic speeds.)[1]

The consistency requirement stated above is stronger than the need to provide the "correct" time to within some specified error bounds, since the inequalities are supposed to be strict. What really matters is not the offset of each clock to true time, but whether the relative offset between any pair of clocks is smaller than the minimum communication delay between the corresponding nodes: if that is achieved, programs will not be able to observe inconsistent timestamps.

The consistency requirement is in fact so strong that it is difficult to guarantee (i.e. prove that it holds, given reasonable constraints on external steering and delay variance). When data integrity depends on it, a separate mechanism is needed to enforce consistency. For example, IBM's Message Time Ordering facility (MTOF [2]) delays delivery of a message (if necessary) until the receiver's clock has caught up with the sender's timestamp. The goal of clock synchronization is then to avoid triggering MTOF (which does have

[1]If the resolution is high enough, relativistic effects can in fact be observed for practical earthbound systems: the Global Positioning System (GPS) takes these effects into account.

an effect on performance) as much as possible.

One solution is for every node to get its time from a single source, using stable delay-compensated links (after an initial tuning sequence, programmable delay lines are adjusted so that timing pulses arrive at each node within microseconds of each other): this is the Sysplex Timer® mechanism used in IBM's zSeries Parallel Sysplex®. This solution is fairly expensive, and does not scale well to geographically distributed clusters.

Another solution would be for each node to be attached to a GPS receiver. For a fixed location, with at least four Global Positioning System satellites in view for a sufficient settling period, microsecond accuracy can be achieved. Unfortunately signal outages are common, and it does not take long for ordinary oscillators to drift by tens of microseconds.

A distributed way to achieve mutual synchronization is to use timestamped message exchanges over the same (or better) links than are used for communication. The usual four-timestamp method of NTP (Network Time Protocol [15]) permits the offset between sender and receiver to be computed, assuming symmetric forward and backward communication delays. One node can then steer its clock to absorb any offset (adjust to its clock source). The literature ([4, 15]) warns sternly against clock dependency loops, however, which is why most synchronization networks use a stratified approach starting from a Primary Reference Clock (called "Stratum-1"), using "Peer" mode at best to obtain a smoother clock in an environment with high link delay variance. Indeed, one can construct pathological cases of clock dependency loops where each clock thinks it is slower than its neighbor (it caught its neighbor during overshoot of a correction phase to react to an earlier perceived slowness), with the net effect that the entire network "takes off" (at least until a saturation point is reached).

Stratified systems require explicit configuration, however, at least with respect to designating the Stratum-1. In order to deal with node failures, a recovery mechanism (typically also preconfigured) must be in place to avoid global failure. In a Peer-to-Peer system, as long as the network remains connected, surviving nodes can still synchronize with each other. Does this resilience always come at the expense of possible instability problems?

In this paper we propose an approach that retains the resilience of a fully distributed peer-to-peer synchronization network with the stability guarantees of a hierarchical synchronization network.

Section 2 describes the hierarchical system STP that serves as our reference. Section 3 presents our alternative, the Almost Peer-to-Peer (AP2P) clock synchronization protocol. Section 4 evaluates the performance of AP2P relative to STP. Related work is discussed in Section 5. Section 6 concludes the paper.

2. Server Time Protocol (STP) – a hierarchical protocol

The successor to IBM's Sysplex Timer® solution is the *Server Time Protocol (STP)* [16], announced in July 2005. It uses a stratified message-based mechanism similar to NTP, using Coupling-Facility links (the links used in a zSeries Parallel Sysplex®). Clock steering is available at the zSeries hardware level, and sophisticated filtering algorithms are used to extract relative clock offset and skew, from which a clock steering rate is derived. Recoverability is achieved by pre-configuring an alternate Stratum-1 server, and enhanced by a so-called "triad" configuration where a third server is designated as an arbiter that can assist in discriminating link failures from node failures, so as to permit a swift Stratum-1 takeover when warranted.

Unlike NTP, the communication paradigm is that of a direct response to a command. In STP, a node periodically exchanges timing packets with each of its neighbors, i.e. the other nodes that it is directly connected to. Each exchange provides a set of four timestamps, the first (A:sent) and last (A:rcvd) derived from the local clock, and the middle two (B:rcvd and B:sent) derived from the remote clock. Round-trip delay and Offset samples are derived from this, but unlike NTP, the reported values are based on filtering applied to a sliding window of recent exchanges, using an algorithm based on the Convex Hull method described in [23]. This also provides a good estimate of the *skew* between nodes A and B.

A clock selection algorithm selects exactly one of the attached servers to be the clock source, taking stratum into account, so as to eliminate clock dependency loops. From the skew and offset relative to the clock source, a node computes a *steering rate adjustment* so as to steer the local clock towards agreement with the clock source.

3. Almost Peer-to-Peer (AP2P) clock synchronization

In contrast to the hierarchical approach of STP, we propose the "Almost Peer-to-Peer" clock synchronization mechanism, which we will call AP2P for short. We assume that each node, n, has a unique numeric ID, ID_n. We also assume that each node knows the set of its neighbors, G_n; i.e., the other nodes that it is directly connected to. A node does *not* need to know the entire network topology. We do however assume that the network is connected.

As in STP, each node periodically exchanges timing packets with its immediate neighbors, from which it obtains the four timestamps and four other items, described below. Offset and Skew are determined by clock filtering, but unlike STP, the steering correction takes all neighbors into account (like Peer-to-Peer), but not necessarily uniformly, and

there is a specific difference from pure Peer-to-Peer (hence "Almost"): A node which considers itself to be the *Leader* does not adjust its clock rate.

Leadership election is therefore a critical component of AP2P. It is however quite different from traditional Leadership Election ([9, 14, 21]) because transient states with no Leader, or with more than one, are benign (as long as they don't last too long). The main difference is the fact that, in AP2P, it is not required that everybody should know that everybody knows the new leader. This greatly reduces the complexity of the algorithm, and completely avoids the non-linear communication overhead in many of the traditional mechanisms.

Synchronization per se (e.g. clock steering, sampling interval selection to achieve desired synchronization accuracy) is not addressed in great detail here, as it would be the same as in STP (see Section 3.2).

3.1. Leader election mechanism

Exactly one of the nodes is the *correct* leader. In the steady-state case, all of the nodes agree on the identity of the unique correct leader. Transient states may exist where either (i) only a subset of the nodes agree on the identity of the unique correct leader, or (ii) the "old" correct leader has failed, and no node has taken over the leadership yet. Note that link failures, which do not disconnect the leader from the rest of the network, do not lead to transient states (see Section 3.3).

In what follows, the t, t', n or p in "time t", "node n", or "packet p" etc. are just labels to denote specific entities, not variables or numeric indices.

A leader plays a role that is similar to a stratum-1 node in STP in the sense that it does not adjust its clock rate based on the timing message exchanges. The other nodes adjust their clock rates in order to remain as synchronized as possible; this is described in Section 3.2.

Let $CL(t)$ denote the correct leader at time t. Each node, n, maintains the following four fields at time t:

1. $L_n(t)$: the ID of the node which n thinks is the leader at time t. Note that n considers itself a leader if and only if $L_n(t) = ID_n$, and n knows the identity of the correct leader if and only if $L_n(t) = CL(t)$.
2. $seq_n(t)$: a sequence number for $L_n(t)$. It indicates how "up-to-date" the leadership information $L_n(t)$ is.
3. $d_n(t)$: the shortest distance (in terms of the number of links) from $L_n(t)$. If n considers itself a leader (i.e., $L_n(t) = ID_n$), then $d_n(t) = 0$. (This field is not used for leader election, but is used in the clock synchronization mechanism explained in Section 3.2).
4. $stamp_n(t)$: the current local timestamp inserted by $L_n(t)$ in its outgoing Timing packets, according to $L_n(t)$'s clock. (This field is not used for leader election, but is used in recovery from node and link failures as will be explained in Section 3.3).

Each timing packet, p, identifies its sender, $sender(p)$, and carries $< L_p, seq_p, d_p, stamp_p >$ which is a copy of the corresponding four-tuple stored at $sender(p)$ at the time the packet is sent. If the sender considers itself to be the leader, it refreshes its $stamp$ from its local Logical Clock before copying it to $stamp_p$.

The initial values of the sequence numbers (i.e., $\forall i \in N, seq_i(t_0)$, where N is the set of nodes and t_0 is the system initialization time) can be either chosen randomly from a certain domain of valid sequence numbers or configured by a system administrator. We assume that initially at least one node considers itself a leader (i.e., $\exists i \in N, L_i(t_0) = ID_i$). A node, i, which does not initially consider itself a leader, sets its $L_i(t_0)$ to ∞ (practically, any value that is guaranteed to be larger than any valid node ID) and its $seq_i(t_0)$ to $-\infty$ (practically, any value that is guaranteed to be smaller than any valid sequence number). Afterwards, $L_i(t)$ and $seq_i(t)$ are updated in *only two* cases: (1) receiving an incoming packet (this case is covered in this section), and (2) recovering from node failures (this case will be covered in Section 3.3).

The correct leader $CL(t)$ at time t is: $CL(t) = L_{i^*}(t)$, where $i^* = arg\ max_{i \in N} seq_i(t)$. In other words, the highest sequence number "wins", and the unique node IDs are used as tie-breakers to assure global uniqueness. Given that nodes are assumed to exchange timing packets on a regular basis, and that each timing packet includes the fixed-size (four-item) information used for leader determination, a simple algorithm permits all nodes to end up agreeing on a common leader from any starting condition that includes at least one leader. There is no specific "election" phase – leadership determination is an ongoing distributed process, so it can quickly react to any changes. Procedure HandleTimingPacket(p) (shown in Figure 1) runs whenever a node n receives a timing packet p from a neighbor $sender(p)$. The node will compute a new four-tuple $< L_n(t'), seq_n(t'), d_n(t'), stamp_n(t') >$ from the current four-tuple $< L_n(t), seq_n(t), d_n(t), stamp_n(t) >$ and the four-tuple included in the packet $< L_p, seq_p, d_p, stamp_p >$ (sent by $sender(p)$).

Figure 1 gives a detailed description for procedure HandleTimingPacket(p). The first part of HandleTimingPacket(p) implements the propagation of leadership information: if the packet's sequence number is larger than the node's current sequence number, or if the numbers are equal but the packet's leader ID is lower than the node's recorded leader ID, the packet's sequence number and leader ID are accepted as the new values to be recorded at this node. It is important to note that a node does *not* voluntarily claim that *another* node is the leader.

```
Procedure HandleTimingPacket(p) {
  /* Part 1 – update Leader info */
  if ( seq_p > seq_n(t) ) { L_n(t') = L_p; seq_n(t') = seq_p; }
  else if ( seq_p == seq_n(t) ) { L_n(t') = min(L_n(t), L_p);
       seq_n(t') = seq_n(t); }
  else { L_n(t') = L_n(t); seq_n(t') = seq_n(t); }

  /* Part 2 – maintain timestamp */
  if ( L_n(t') == ID_n ) { stamp_n(t') = LogicalClock_n(t');
       d_n(t') = 0; }
  else if ( L_n(t') == sender(p) ) { stamp_n(t') = stamp_p;
       d_n(t') = 1; }
  else if ( L_n(t') == L_p ) {
    if ( L_n(t') == L_n(t) ) {
      if ( stamp_p ≥ stamp_n(t) ) { stamp_n(t') = stamp_p;
        d_n(t') = min(d_n(t), d_p + 1)); }
    } else { stamp_n(t') = stamp_p; d_n(t') = d_p + 1; }
  } else { stamp_n(t') = stamp_n(t); d_n(t') = d_n(t); }
}
```

Figure 1. Handling a Timing packet p by node n.

Theorem 1: *All the nodes in the network will eventually agree on the identity of the unique correct leader.*

Proof: Define $f(t)$ as the number of nodes whose leader ID is equal to the identity of the correct leader at time t. We shall prove that this function is non-decreasing and will reach $|N|$, the number of nodes in the timing network. Specifically, $f(t) = |\{ i \in N, L_i(t) = CL(t) \}|$.

Note that $1 \leq f(t) \leq |N|$ (we assumed that initially at least one node considers itself a leader). Exactly one of those nodes that initially consider themselves as leaders (namely the one with the largest sequence number and, in case of ties, smallest leader ID) is the correct leader; hence, initially there exists exactly one node, $i^* \in N$, such that $L_{i^*}(t_0) = ID_{i^*} = CL(t_0)$.

Every timing packet reception either increases $f(t)$ or keeps it constant. To see why, consider the following two cases for a node n that has sent Timing Request packets to all its neighbors, G_n, at time t and has received Timing Response packets from all of them at time t':

Case A: If neither n nor any of its neighbors has its leader ID set to $CL(t)$, then neither n nor any of its neighbors will discover the identity of the correct leader after the Timing message exchange. Hence, $f(t)$ remains constant, i.e. $f(t') = f(t)$.

Case B: If either n or at least one of its neighbors has its leader ID set to $CL(t)$, then there are two subcases:

Case B-1: If $L_n(t) = CL(t)$, and k of n's neighbors do not know the identity of the correct leader, then all of the k neighbors will set their leader IDs to $CL(t)$ after they receive the Timing Request packets from n; hence, $f(t') = f(t) + k$.

Case B-2: If $L_n(t) \neq CL(t)$, and at least one of n's neighbors has its leader ID set to $CL(t)$, then n will set its leader ID to $CL(t)$ after it receives the Timing Response packet from that neighbor; hence, $f(t') = f(t) + 1$.

Assuming that each node gets a chance to participate in the leader election mechanism (this assumption is reasonable because the leadership information is carried in the Timing packets that nodes are exchanging periodically in order to achieve clock synchronization), this ensures that neither Case A nor Case B-1 with k = 0 will be the case forever; hence, $f(t)$ will increase until it eventually reaches $|N|$. $f(t) = |N|$ means that $\forall i \in N, L_i(t) = CL(t)$. Hence, after $f(t) = |N|$, neither Case A nor Case B-2 may happen. The only possible case will be Case B-1 with $k = 0$ (because all of n's neighbors already know the identity of the correct leader). Therefore, once $f(t)$ reaches $|N|$, $f(t)$ will remain constant. This completes the proof. □

It should be mentioned that using sequence numbers gives system administrators the ability to pre-determine the leader of a network (e.g., because this node has access to a good external time reference). A system administrator simply needs to assign this node a sequence number that is strictly larger than the sequence number of any other node in the network, and configure this node to initially consider itself a leader. (We make use of this nice feature in some of our experimental results in Section 4.) Similarly, in order to prohibit a node from being the leader of a network, a system administrator simply needs to assign this node a sequence number that is strictly smaller than the sequence number of at least one other node in the network.

To accelerate propagation of leadership change, a node that just updated its recorded Leader ID will immediately send a LEADER packet to each of its neighbors (instead of waiting for the next scheduled timing exchange). Such a packet p contains only its sender ID and the four leadership information fields: $< L_p, seq_p, d_p, stamp_p >$. It is processed just like any other packet with regard to this information. System Initialization time counts as a change in Leadership for those nodes that initially consider themselves to be a leader (there is at least one).

Theorem 2: *Regardless of how many nodes initially declare themselves as leaders, all the nodes in the network will agree on the identity of the unique correct leader after delay $D \times P$ from the system initialization time t_0, where D is the maximum shortest distance (in terms of the number of links) from the correct leader to any node, and P is the maximum propagation delay of a link.*

Proof: Recall that, regardless of how many nodes initially declare themselves as leaders, exactly one of them

(namely the one with the largest sequence number and, in case of ties, smallest leader ID) is the correct leader. We only need to consider the LEADER packets sent by this correct leader (identified as i^* below).

After delay P from the time i^* sends the LEADER packet, all of the nodes that are direct neighbors of (i.e., one link away from) i^* will have received the LEADER packet. All of these direct neighbors will accept the leadership information contained in the LEADER packet. This is because i^* has a larger sequence number (or, in case of ties, a smaller leader ID) than that of any other node in the network (that is the definition of the correct leader). Furthermore, for each node $j \in G_{i^*}$, node j's leader ID will change after handling the LEADER packet. This is because j has no other way of previously knowing that i^* is a leader (recall that no node voluntarily claims that another node is the leader). Hence, node j will forward a LEADER packet to each of its neighbors.

After $2 \times P$ from the time i^* sends the LEADER packet, a similar argument can be stated for all the nodes whose shortest distance (in terms of the number of links) from i^* is 2. In general, after $d \times P$, all the nodes whose shortest distance from i^* is d will agree on the identity of the correct leader i^*. Hence, if D is the maximum shortest distance from i^* to any node, all the nodes in the network will agree on the identity of the unique correct leader after $D \times P$. This completes the proof. □

Note that at $t_0 + D \times P, \forall n \in N, L_n(t_0 + D \times P) = i^*$; hence, the forwarding of LEADER packets will stop because no node will have its leader ID changed after handling a LEADER packet. In fact, it is easy to see that each node will send a LEADER packet, declaring i^* as a leader, to each of its neighbors exactly once. Hence, the overhead caused by broadcasting LEADER packets is insignificant.

It should also be noted that if LEADER packets are lost, agreement on the identity of the unique correct leader will only be delayed, but will eventually be achieved (as we proved in Theorem 1) because leadership information is carried in all Timing packets that are exchanged between the nodes.

3.2. Clock synchronization mechanism

Node n sends a Timing Request packet to each of its neighbors at regular intervals τ.[2] Upon receiving a Timing Response packet from a neighbor, n runs the convex hull filtering algorithm to compute a suggested change in its steering rate, and records it in a small history array. It then computes the total change in its steering rate as a weighted average of the recent steering rate changes computed for its neighbors. (If a neighbor does not reply after a reasonable timeout, e.g. three times the estimated round-trip delay (available from the filter computation), the steering correction can be computed from the remaining information, and the age of the current leadership information can be checked.)

The weight assigned to each suggested steering rate change depends on the distance d to leader (reported as d_p in a Timing Response packet p), and on whether the reported Leader ID L_p agrees with the node's own view thereof, L_n: if not equal, a weight of zero is assigned (the information is not believed), otherwise a weight of b^{-d} is assigned, where the base $b \geq 1$ can be tuned to control the ratio between the weight assigned to a closer node to that assigned to a further node. In fact, an exponentially-weighted moving average is used for each neighbor, so that more recent steering suggestions have more effect than older ones.

3.3. Recovery from node and link failures

The most important type of node failure is the failure of the current leader. In this case, another node has to take over the leadership by becoming the new leader. Furthermore, it would be better if one of the nodes that were direct neighbors of the "old" leader became the new leader: such a node is most likely better synchronized with that leader's clock than nodes that are further away. This preference is not absolute, however, since we would like to handle the case where a dead leader's direct neighbors fail before having assumed leadership and propagated that information. Instead, any node can be the new leader, with the nodes that were closer to the old leader having a better chance of being the new leader.

Similarly, the most important type of link failure is the failure of a link that is connected to the current leader (but not the last such link – we assume there is enough link redundancy so that the timing network remains connected). In this case, we do not want a leadership change because the current leader is still operating and did not fail.

In summary, we need a mechanism that discovers a leader's failure and differentiates between a node failure and a link failure. This is where the leader timestamps recorded at each node ($stamp_n$) and transmitted in each packet ($stamp_p$) come into play. Recall that this timestamp is updated whenever a node that considers itself to be the Leader sends out a Timing packet (Request or Response).

Now is the time to examine the second part of HandleTimingPacket(p) (Figure 1). When n is a direct neighbor of the leader, it accepts the new timestamp, which is guaranteed to be more up-to-date than that stored at the node, because it comes from the leader itself. Otherwise, if n is not a direct neighbor of the leader, it needs first to check

[2] Recall that in STP, a node exchanges Timing packets with each of its neighbors too. Therefore, given the same network topology and the same outgoing message interval τ, the number of Timing packets in AP2P is the same as that of STP.

whether the packet timestamp is more up-to-date than its own. This check is only required if n did not change its leader ID – if not, the source clocks are not comparable, and the new timestamp should be accepted unconditionally (it might be from a node about to become a new leader).

It is now easy to see that if a link that is connected to the current leader failed, the leader timestamps can still be propagated in the network as long as the current leader is still connected to the network. We use these timestamps to detect the current leader's failure and trigger a leadership change: If the leader timestamp, $stamp_n(t)$, is not refreshed for $d_n(t) \times T$ (where T is a parameter of the recovery mechanism), n considers the current leader to have failed, and declares itself as a leader. Specifically, n sets its leader ID $L_n(t)$ to its own ID ID_n, its $d_n(t)$ to 0, its $stamp_n(t)$ to its local logical clock, and increments its $seq_n(t)$. Incrementing $seq_n(t)$ is required so that nodes accept the new leader's information and discard that of the old leader. Furthermore, n broadcasts a LEADER packet declaring itself as a leader, as described in Section 3.1.

It should be noted that multiple nodes may detect the old leader's failure (almost) simultaneously and declare themselves as new leaders. In this case, the conflict will be resolved by the leader election mechanism explained in Section 3.1. As we proved in Theorems 1 and 2, this mechanism guarantees that all the nodes in the network will eventually agree, within a finite time, on the identity of a new unique correct leader.

4. Performance evaluation results

The performance evaluation was carried out using the J-Sim network simulator [12]. For the most part we use the traditional measure of maximum offset from a common reference, but we do include an example of almost-perfect synchronization, where MTOF could induce small extra delays during sharp steering events.

Different network topologies were used in the experiments. The link delays follow different distributions including Pareto, log-Normal and Exponential distributions. We have observed very similar patterns for different distributions. In this paper, we will only present results for Pareto link delay distributions, with parameter k and minimum value $10\mu s$. Smaller values of k correspond to links with larger delay variations. Details about all the other distributions can be found in a research report [19].

The most severe challenge to maintaining synchronization is when the Leader changes its clock rate – e.g. to track some external time reference. It may take a few seconds for the network to adjust – we call this the "steering phase" of the reaction (as opposed to the "normal phase").

4.1. Clock synchronization accuracy

We use the maximum deviation between a clock and the leader's clock as the measure for the synchronization accuracy. Because each clock in the AP2P mechanism is influenced by other neighboring clocks who may have less up-to-date information from the leader, the synchronization accuracy of AP2P may be worse compared with a hierarchical approach such as STP.

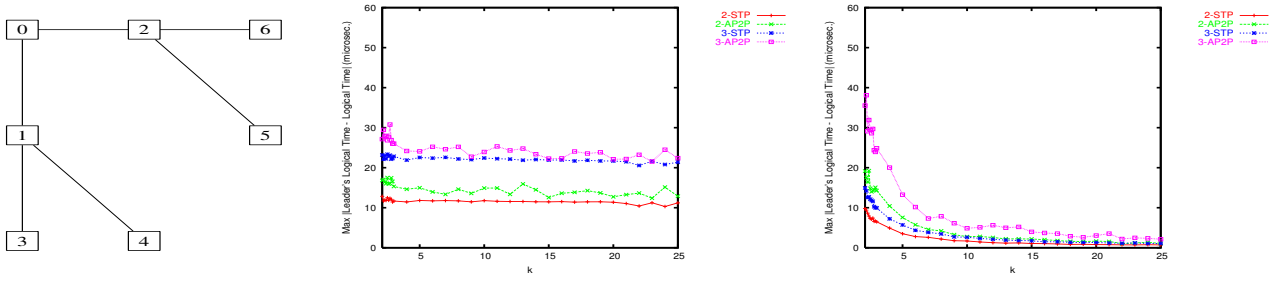
We first consider the network of a tree topology with three strata, as shown in Figure 2-(a). Node 0 (the stratum-1 node in the STP case) starts as the leader. This is achieved by initially assigning node 0 the largest sequence number in the network, and making node 0 declare itself as a leader. It changes its steering rate three times: from 0 ppm to 25 ppm at time 50 second, to -25 ppm at time 100 second and to 0 ppm at time 150 second. A node exchanges 16 messages per second with each of its neighbors. The steering phase (see above) starts whenever node 0 changes its steering rate, and lasts for five seconds thereafter.

Figure 2 shows the maximum deviation from the leader's clock for stratum 2 and 3 nodes. The horizontal axis k corresponds to the shape parameter for the Pareto distribution. Each data point is the average of 10 simulation runs, using a weight-decay base $b = 2$ (see Section 3.2). We observe that the synchronization accuracy degrades for smaller k, which corresponds to more variable link delays. Furthermore, in normal operation phase, the accuracy is often within the average link delay. In the steering phase, the accuracy is roughly d times the average link delay for stratum-$(d + 1)$ nodes. This corresponds to the propagation delay of the steering information from node 0 to the stratum 2 and 3 nodes. We observe that AP2P does not perform quite as well as STP, but not by much. In later experiments, we show that increasing the value of b resolves this difference in performance between STP and AP2P.

4.2. Clock dependency loops

To evaluate the performance of various clock synchronization mechanisms for more complex network topologies with dependency loops, we first compare the performance of AP2P with that of a purely peer-to-peer clock synchronization protocol (which we call P2P for short). In P2P, there is no leader, and each node assigns an equal weight to each of its neighbors (base $b = 1$). We consider a set of network topologies; each of which is a 2-D torus of $|N|$ nodes. In such networks, as $|N|$ increases, the maximum shortest distance (in terms of the number of links) between two nodes increases but the number of neighbors of a node remains constant.

A node, l, is chosen uniformly at random to be the leader node in the case of AP2P. In both P2P and AP2P, we mea-



(a) A tree of 7 nodes.

(b) Steering phase.

(c) Normal operation phase.

Figure 2. Synchronization accuracy for STP and AP2P.

Table 1. Properties of the network topologies generated using GT-ITM.

Class	Number of Nodes	Number of Links	Maximum Node Degree	Average Node Degree	Network Diameter
A	15	31	8	4.13	4
A	30	64	12	4.27	5
A	45	97	18	4.31	5
A	60	142	20	4.73	5
A	75	182	22	4.85	6
A	90	225	22	5.0	6
B	44	60	6	2.73	9
B	84	195	10	4.64	10
B	124	384	12	6.19	10
B	164	678	14	8.27	8
B	204	1033	17	10.13	8

sure the maximum deviation of the logical clocks of all nodes from the logical clock of node l. As shown in Figure 3, the synchronization accuracy for P2P is almost two times worse than AP2P. This result demonstrates the significant benefit for the leader election mechanism.

Next, we compare the performance of AP2P and STP for larger size networks. We use the GT-ITM network topology generator [22] to generate more realistic networks. We consider two types of networks: non-hierarchical (which we call Class A), and hierarchical (which we call Class B). Table 1 shows the properties of these network topologies. The node degree is the number of neighbors a node has. The average node degree is $\frac{2\times|L|}{|N|}$, where $|L|$ is the number of links and $|N|$ is the number of nodes in the topology. The network diameter is the maximum shortest distance (in terms of the number of links) between any two nodes. Figure 4 shows the 204-node Class B network topology.

Figure 3 shows the maximum deviation from the leader's clock for the stratum-7 nodes for the Class B network topologies in both the steering and normal operation phases.

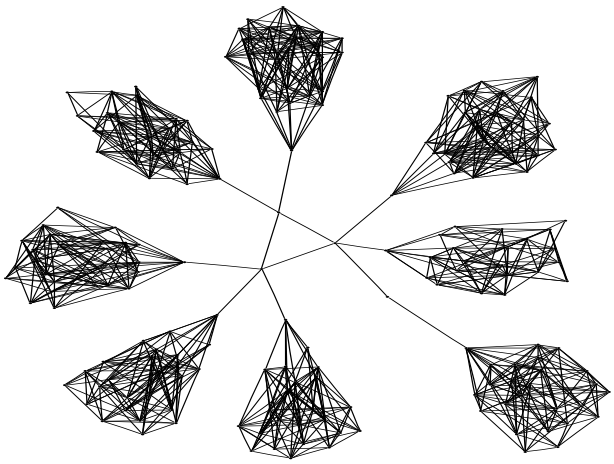
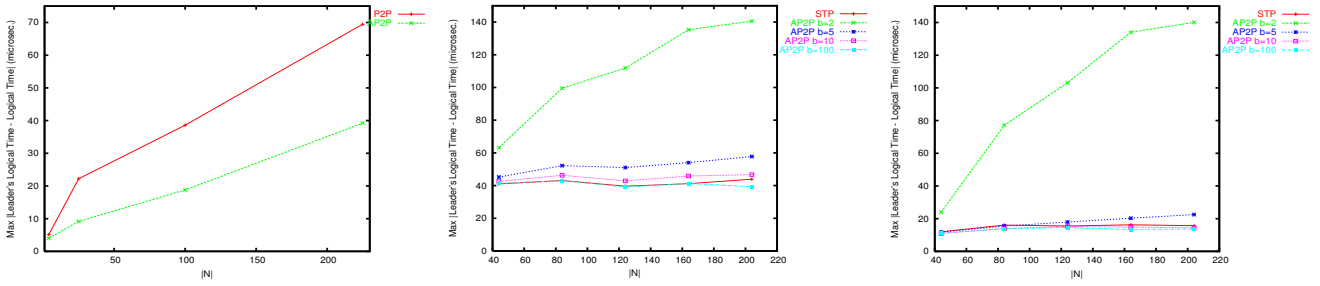


Figure 4. 204-node Class B network topology.

The performance of AP2P with $b = 2$ is considerably worse than that of STP but, as b increases, the effect of neighbors further away from the leader diminishes, and accuracy improves. In particular, in the steering phase, the performance of AP2P with $b = 100$ is very close to that of STP, and in the normal operation phase it is already indistinguishable from that of STP for $b = 10$. Similar results were obtained at the other strata and for Class A networks. This result justifies the need for a weight assignment mechanism that strongly favors neighbors that are closer to the leader.

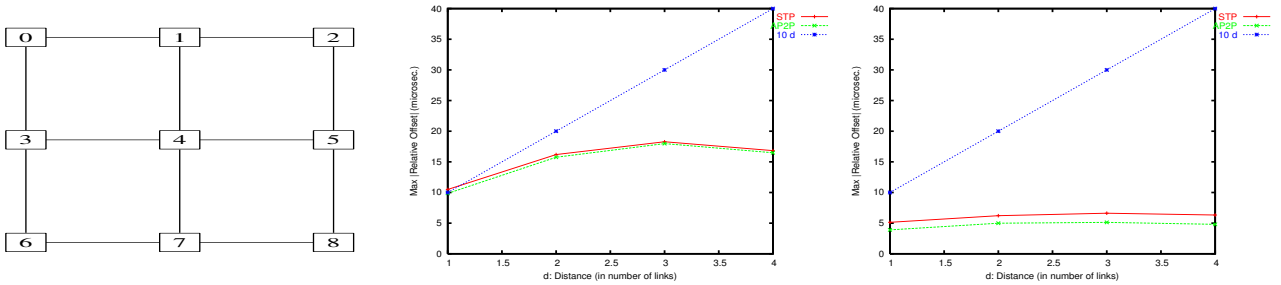
4.3. Relative offset between pairs of clocks

We now study the maximum absolute relative offset between a pair of clocks versus the minimum communication delay between the corresponding nodes. The network topology is a grid of 9 nodes. A node, l, is chosen uniformly at random to be the stratum-1 (or leader) node for STP (or



(a) Torus: P2P vs AP2P (Normal). (b) Class B: AP2P vs STP (Steering). (c) Class B: AP2P vs STP (Normal).

Figure 3. P2P, AP2P and STP accuracy comparisons.



(a) A 3×3 Grid Network. (b) Steering phase. (c) Normal operation phase.

Figure 5. Relative offset versus shortest distance between clocks.

AP2P).

Figure 5 shows the maximum absolute relative offset between a pair of clocks vs. the shortest distance (in number of links) between the corresponding nodes. As shown in Figure 5-(c), in the normal operation phase, the maximum absolute relative offset between a pair of clocks, whose nodes are at a shortest distance of d links from each other, is less than $d * 10\mu$s. In the steering phase (Figure 5-(b)), this is satisfied except for some immediate neighbors $d = 1$, where AP2P just barely manages to avoid MTOF delays, and STP is likely to incur occasional MTOF delays on the order of 1 μs at the onset of external steering. (The moral is to avoid abrupt external steering changes – the desired effect can usually be achieved by a more gradual approach.)

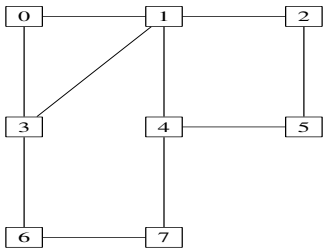
4.4. Failure recovery

We compare the performance of STP with that of AP2P in terms of recovery from node failures. The link failure cases have same types of behavior. We consider the case of two consecutive stratum-1 (or leader) node failures. The network topology consists of eight nodes, as shown in Figure 6-(a), with a link between node 1 (the alternate stratum-1 server in STP) and node 3 (the arbiter server in STP) as required by the triad configuration in STP. At time 50 sec., node 0 fails, causing another node to become the new stratum-1 (or leader) node, as shown in Figure 6-(b) and Figure 6-(c). At time 100 sec., the new stratum-1 (or leader) node fails. In the case of STP, the remaining nodes are not able to maintain synchronization, as shown in Figure 6-(d), although the network is still connected. In fact, the triad configuration in STP *cannot* handle this type of two consecutive stratum-1 node failures. In contrast, the leader election mechanism in AP2P enabled the remaining nodes to elect a new leader and continue to maintain synchronization, as shown in Figure 6-(e).

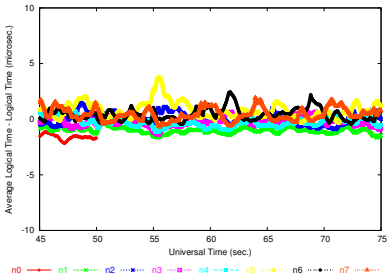
5. Related work

The clock synchronization problem has been extensively studied before (e.g., [5, 6, 10, 15, 17, 18, 20]).

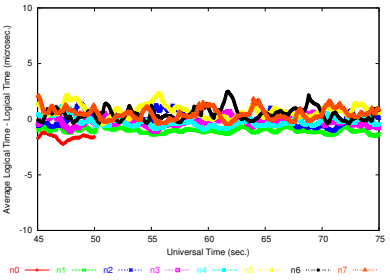
The Network Time Protocol (NTP) [15] is commonly used on the Internet to enable the time of a computer to be synchronized to another server or reference time source. NTP has four modes of operation. In the client/server mode, a client announces its willingness to be synchronized by, but not to synchronize a peer, and sends periodic messages to



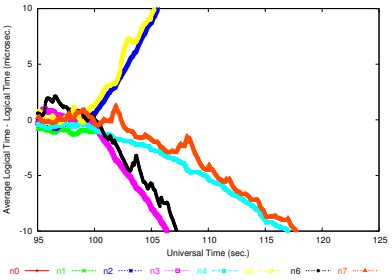
(a) A graph of 8 nodes in a triad configuration.



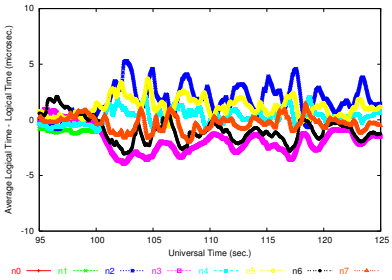
(b) STP: Stratum-1 node (node 0) fails at time 50 sec.



(c) AP2P: Leader node (node 0) fails at time 50 sec.



(d) STP: New stratum-1 node (node 1) fails at time 100 sec.



(e) AP2P: New leader node (node 1) fails at time 100 sec.

Figure 6. STP vs AP2P (b=100) for two consecutive stratum-1 (or leader) node failures.

that peer, while a server announces its willingness to synchronize, but not to be synchronized by its peer, and replies to incoming client request messages. This mode of operation of NTP is similar to STP. In the symmetric active mode, a node announces its willingness to synchronize and be synchronized by a peer, and sends periodic messages to that peer. In the symmetric passive mode, a node announces its willingness to synchronize and be synchronized by a peer, and replies to request messages received from a peer operating in the symmetric active mode. The symmetric modes are similar to AP2P, with the major distinction that the synchronization topology of an NTP-peer subnet must avoid clock-dependency loops, whereas AP2P has no such restriction.

Cristian's probabilistic method for synchronizing clocks [6] uses a strict hierarchical client/server approach, in which a client polls a server for its time. By measuring the round-trip delay of a message, the client can synchronize its clock if the round-trip delays between the client and the server are sufficiently short compared with the required accuracy. Although Olson and Shin [17] present a probabilistic clock synchronization scheme that considers message delay uncertainties and does not depend on a client/server hierarchy, they do not consider fault tolerance and require that the system be organized into a number of overlapping *synchronization groups* where the groups are chosen so that there is a cycle which goes through each node. This requirement may not allow for arbitrary network topologies.

In [5], Attiya et al. present clock synchronization algorithms for a large family of delay assumptions, but they do not consider fault tolerance and assume that clocks are perfect (i.e., do not drift). Although Ostrovsky and Patt-Shamir [18] consider both message delay uncertainties and clock drift, they do no not consider fault tolerance.

In [10], Gurewitz et al. propose the Classless Time Protocol (CTP), which is also a non-hierarchical peer-to-peer approach for clock synchronization. In CTP, one node is distinguished as a "reference time node", and the goal is to synchronize the nodes in the network with that reference node. This is achieved by having each node send and receive probe packets (NTP packets) to and from its neighbors and adjust its clock accordingly. The clock synchronization problem is then formulated as an optimization problem, whose input includes all the delay measurements, and whose output is the set of clock adjustments for all the

nodes. Clock filtering is better than NTP, but CTP still only considers offset correction, and the analysis assumes there is no skew. In contrast, AP2P takes skew into account, which leads to better synchronization.

Another peer-to-peer clock synchronization algorithm is presented in [20]. The algorithm applies Bayesian estimation of time errors, clock rate errors, and the communication delay between the synchronizing nodes. A (virtual) tree network topology is required, and the actual round-trip time of a message is assumed to have a Gaussian distribution (which may not be satisfied by most network connections).

Providing fault-tolerant clock synchronization is not new. It has been discussed before in several papers (e.g., [7, 8,13]). Compared to a purely hierarchical (client/server) solution such as STP, AP2P achieves superior fault tolerance while maintaining the high synchronization accuracy.

6. Conclusion

We have presented AP2P, a novel clock synchronization protocol. AP2P operates in an almost peer-to-peer manner that combines the high synchronization accuracy of a purely hierarchical (client/server) clock synchronization protocol, and the robust fault-tolerance of a purely peer-to-peer one. In AP2P, a unique node is elected as a leader in a distributed manner. Each non-leader node adjusts its clock steering rate based on message exchanges with its neighbors. AP2P makes use of a weight assignment mechanism that gives neighbors that are closer to the leader more effect on the clock adjustment than those that are further away from the leader.

Simulation results, which have been conducted on several network topologies, have shown that AP2P, using an appropriate weight assignment mechanism, can provide a clock synchronization accuracy that is indistinguishable from that of STP, a purely hierarchical protocol.

Commercial hierarchical timing networks (e.g., STP) do provide fault tolerance, but at the cost of a complicated pre-configured recovery mechanism, whereas in AP2P, the recovery mechanism (namely, electing a new leader) does not require pre-configuration. Furthermore, AP2P is more fault-tolerant than STP because it can recover from certain types of multiple failures that STP cannot recover from.

References

[1] Task 4–Using Syslog, NTP, and Modem Call Records to Isolate and Troubleshoot Faults. http://www.cisco.com/univercd/cc/td/doc/cisintwk/intsolns/dialsol/nmssol/syslog.htm.

[2] Method and system for providing a message-time-ordering facility. http://www.freshpatents.com/Method-and-system-for-providing-a-message-time-ordering-facility-dt20041118ptan20040230854.php.

[3] IEEE 802.11 Standard. Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, 1999.

[4] J. E. Abate, E. W. Butterline, R. A. Carley, P. Greendyk, A. M. Montenegro, C. D. Near, S. H. Richman, and G. P. Zampetti. AT&T's new approach to the synchronization of telecommunication networks. *IEEE Communications Magazine*, 27(4):35–45, April 1989.

[5] H. Attiya, A. Herzberg, and S. Rajsbaum. Optimal clock synchronization under different delay assumptions. *SIAM Journal on Computing*, 25(2):369–389, 1996.

[6] F. Cristian. Probabilistic clock synchronization. *Distributed Computing*, 3(3):146–158, 1989.

[7] D. Dolev, J. Y. Halpern, B. Simons, and R. Strong. Dynamic fault-tolerant clock synchronization. *J. ACM*, 42(1):143–185, 1995.

[8] S. Dolev and J. L. Welch. Self-stabilizing clock synchronization in the presence of Byzantine faults. *J. ACM*, 51(5):780–799, 2004.

[9] H. Garcia-Molina. Elections in a distributed computing system. *IEEE Transactions on Computers*, January 1982.

[10] O. Gurewitz, I. Cidon, and M. Sidi. Network time synchronization using clock offset optimization. In *Proc. of IEEE ICNP'03*.

[11] C. Intanagonwiwat, R. Govindan, D. Estrin, J. Heidemann, and F. Silva. Directed diffusion for wireless sensor networking. *IEEE/ACM Transactions on Networking*, 11(1):2–16, February 2003.

[12] J-Sim. http://www.j-sim.org/.

[13] L. Lamport and P. M. Melliar-Smith. Synchronizing clocks in the presence of faults. *J. ACM*, 32(1):52–78, 1985.

[14] D. Malkhi, F. Oprea, and L. Zhou. Omega meets Paxos: Leader election and stability without eventual timely links. In *Proc. of DISC'05*.

[15] D. L. Mills. Network Time Protocol (version 3) specification, implementation and analysis. Network Working Group, RFC 1305, March 1992.

[16] B. Ogden, J. Fadel, and B. White. IBM System z9 109 Technical Introduction. IBM Redbooks SG24-6669, July 2005.

[17] A. Olson and K. G. Shin. Probabilistic clock synchronization in large distributed systems. *IEEE Transactions on Computers*, 43(9):1106–1112, 1994.

[18] R. Ostrovsky and B. Patt-Shamir. Optimal and efficient clock synchronization under drifting clocks. In *Proc. of ACM PODC'99*.

[19] A. Sobeih, M. Hack, Z. Li, and Z. Liu. Almost peer-to-peer clock synchronization. IBM Research Report, 2007.

[20] M. W. Soijer. Bayesian peer-to-peer clock synchronization for accurate interval measurements. Unpublished. http://soijer.de/features/clock/index.html.

[21] S. D. Stoller. Leader election in distributed systems with crash failures. Technical Report TR481, Indiana University, July 1997.

[22] E. W. Zegura, K. L. Calvert, and S. Bhattacharjee. How to model an internetwork. In *Proc. of IEEE INFOCOM'96*.

[23] L. Zhang, Z. Liu, and C. Xia. Clock synchronization algorithms for network measurements. In *Proc. of IEEE INFOCOM'02*.