I. Impact of synchronization

In asynchronous network, if I do not receive a message I cannot conclude anything.
In a synchronous network, if I do not receive a message at time t, I know it was not sent at time t-1.
II. Synchronizers

Suppose we have a synchronous algorithm.

In this algorithm, messages are sent at rounds 0,1,2,...

A message sent from P to Q at round t (of P) will be received before round t+1 (of Q).

(a stronger assumption will be that all clocks are synchronized)

We want to run it on an asynchronous network

Synchronous algorithm

+ 

Synchronizer

= 

Asynchronous algorithm
I. General network

There exists a clock at each processor, that generates pulses.

M sent from processor P to processor Q at pulse t (of P) arrives at Q before its t+1st pulse

Given a synchronous protocol, we first add to it a mechanism of acknowledgements:
whenever Q receives a message from P it first sends an acknowledgement to it, and then proceeds with its other operation.

Note: this will double the overall message complexity

But: in many protocols a form of acknowledgements already exists - implicitly or explicitly (e.g., PIF)

Eventually a node knows when all of its messages, sent in its current pulse t, have been received by its neighbors.

We term such a node safe.

In order to start pulse t+1, it is not sufficient for a node to be safe: it must make sure that all the messages sent to it by its neighbors at their pulse t have been received.

If a node learns that all of its neighbors are safe, it can move to its next pulse.

It thus remains to ensure a mechanism that will let a node know that all of its neighbors are safe.
We describe two such synchronizers.

1: local, highly-distributed, where pulses are generated fast, but cost a lot of messages.

2: global, centralized, where pulses are generated slowly, which results in saving of messages.

Synchronizer $\alpha$

When a node knows that all of its messages of the current pulse have been received, it notifies all of its neighbors that it is safe.

When a node receives safe messages from all of its neighbors, it moves to its next pulse.

Synchronizer $\alpha$ has the property of being expensive in messages - it uses $O(|E|)$ messages per pulse - and cheap in time - clearly $O(1)$, on a network $G = (V,E)$.
Synchronizer $\beta$

Given a rooted spanning tree in the network. Each node knows its parent and children in the tree. The root initiates a pulse, and sends it down the tree. When a node learns that it is safe and all of its children are safe, it sends a safe message to its parent. (this process starts at the leaves). When the root learns that all the nodes are safe, it initiates the next pulse, etc

Synchronizer $\beta$ has the property of being expensive in time - it takes $O(|V|)$ units of time for generating each pulse - and cheap in messages - only $O(|V|)$ messages are sent per each pulse, on a network $G = (V,E)$.

2. Bounded delay network

Assumption: a message is delivered within a delay $d < 1$.

Recall: we have a synchronous algorithm. In this algorithm, messages are sent at rounds $0,1,2,...$ A message sent from $P$ to $Q$ at round $t$ (of $P$) will be received before round $t+1$ (of $Q$).

We design a synchronizer for this network.
Initially, nodes set their local timer to 0, using PI.

Denote by $t_p$ the global time when node $P$ sets its timer to 0.

the actual values of $t_p$ is not known to the processors

By the nature of the PI algorithm:

$t_{p} - 1 < t_{q} < t_{p} + 1$

for every two neighboring nodes $P$ and $Q$.

Each node, upon receiving the first init message, resets its timer to 0 and starts counting time.

Its time slot $m$ will begin at time $m\tau$ and end before time $(m+1)\tau$ for $m \geq 0$.

$\tau$ will be determined later

$0 \quad \tau \quad 2\tau \quad m\tau \quad (m+1)\tau$
Message sent by $P$ at pulse $m$ is received by $q$ at time $t$.

Send messages at start of a time slot

The message of phase $m$ of $P$ will be sent at the beginning of its $m$-th time slot.
That is, at time $t_p + m\tau$

Message sent at time $t_q + m\tau$

Recall: message sent by $P$ at pulse $m$ is received by $q$ at time $t$

\[
\begin{align*}
t &< t_p + m\tau + 1 \\
< t_q + m\tau + 2 = \\
t_q + (m+1)\tau + (2 - \tau)
\end{align*}
\]
\[ t < t_q + (m + 1)\tau + (2 - \tau) \]
We want \[ t < t_q + (m + 1)\tau \]

Choose: \( \tau = 2 \)

Namely, send messages at times 0, 2, 4, 6, ...
(rather than 0, 1, 2, 3, ...

However...
\[ t \geq t_p + m\tau \]
\[ > t_q + m\tau - 1 = t_q + (m - 1)\tau + (\tau - 1) \]
Namely, message received by \( q \) after the start of time slot \( m-1 \)
\[
(m-1)\tau \quad m\tau \quad (m+1)\tau
\]

If we choose \( \tau = 2 \) - namely, we send the messages at times 0, 2, 4, 6, ...
(rather than 0, 1, 2, 3, ...), then we have to add one bit to each message (0, 1, 0, 1, ...), so that we know how exactly to simulate the synchronous algorithm.

If we do not want to use this extra bit...
Send messages after delay

Message sent at time \( t_p + m\tau + \phi \)

Recall: message sent by \( P \) at pulse \( m \) is received by \( q \) at time \( t \)

\[
\begin{align*}
t &< t_p + m\tau + 1 + \phi < \\
t_q + m\tau + 2 + \phi &= \\
t_q + (m+1)\tau + (2 - \tau) + \phi
\end{align*}
\]

\[
\begin{align*}
t \geq t_p + m\tau + \phi &> \\
t_q + m\tau - 1 + \phi &= \\
t_q + (m-1)\tau + (\tau - 1) + \phi
\end{align*}
\]

\[
\begin{align*}
t &< t_q + (m+1)\tau + (2 - \tau) + \phi \\
t > t_q + (m-1)\tau + (\tau - 1) + \phi
\end{align*}
\]

Choose: \( \tau = 3, \ \phi = 1 \)

\[
\begin{align*}
t &< t_q + (m+1)\tau + (2 - 3) + 1 = t_q + (m+1)3 \\
t > t_q + (m-1)\tau + (3 - 1) + 1 = t_q + 3m
\end{align*}
\]
III. Communication-time trade-off

In a synchronous system we can trade time for messages.

In asynchronous network, if I do not receive a message I cannot conclude anything.
In a synchronous network, if I do not receive a message at time $t$, I know it was not sent at time $t-1$.

Recall: Example 1

4 days, 2 professor

Theorem:

In a synchronous network, where id's are integers $>0$, $n$ known, and smallest id is $I$,
We can find smallest id in time $O(b^{\frac{n}{2}})$ using $O(bn)$ bits, for any integer $b > 0$. 
decide(x)
  clock := 0;
  if my_id > x
    then state := undecided
    else { send "yes";
           state := decided }
  start counting;
  if "yes" is received with clock ≤ n:
    if state = undecided
      then { send the message;
             state := decided; }
    else ignore the message:

If all identities are > x, then all processors become "undecided" and 0 bits are sent.
Otherwise, all processors become "decided", and n bits are sent.

(this is for a unidirectional ring of size n)
Find the minimum, knowing it is between 1 and 7, using at most 3n bits, and in time at most 4n.

**1st problem: Given t, and b, determine N.**

\[ f(t,b) : \]

largest integer s.t. a minimum in the range 1..f(t,b) can be found using at most t questions and b overestimates.

\[ f(t,1) = t + 1 \quad \forall t \geq 1 \]
\[ f(t,t) = 2^t \quad \forall t \geq 1 \]
\[ f(t,b) = f(t-1,b) + f(t-1,b-1) \quad \forall t > b \geq 1 \]
\[
f(t, b) = \binom{t}{0} + \binom{t}{1} + \binom{t}{2} + \cdots \binom{t}{b}
\]

Proof: by induction.

\[f(2, 1) = 3\]
\[f(2, 2) = 4\]
f(3, 2) = 3 + 4 = 7

Optimal and unique tree.
Proof: by induction

Alternative proof:
Binary tree, leaves at distance at most and at most b steps to the left
\{\mathbf{\varepsilon}\} with no steps to the left
\{1\} with 1 step to the left
\{2\} with 2 steps to the left
...
\{b\} with b steps to the left
1st problem: Given \( t \), and \( b \), determine \( N \).

2nd problem: Given \( N \) and \( b \), determine \( t \).

\[
h(N,b) = \min_{t} \{ \{ f(t,b) \geq N \} = \min_{t} \left\{ \sum_{j=0}^{b} \binom{t}{j} \geq N \right\}
\]

Lemma: \( \exists \ -1 \leq (N,b) \leq 1 \) s.t.

\[
h(N,b) = (b! \cdot N)^{1/5} + \varepsilon(N,b)
\]

Proof: hw.

1st problem: Given \( t \), and \( b \), determine \( N \).

2nd problem: Given \( N \) and \( b \), determine \( i \).

3rd problem: Given \( b \), determine \( i \).
Theorem: The minimum $i$ can be determined using at most $O(bn)$ bits and $O(bn^{1.5})$ time for any integer $b>0$. 

<table>
<thead>
<tr>
<th>$t$</th>
<th>b</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>7</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>11</td>
<td>15</td>
<td>16</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>16</td>
<td>26</td>
<td>31</td>
<td>32</td>
<td></td>
</tr>
</tbody>
</table>

$t$: Number of questions, $b$: Error bound.
References

B. Awerbuch,
An efficient network synchronization protocol
16th STOC, 1984, pp.

References

C.T. Chou, I. Cidon, I. Gopal and S. Zaks,
Synchronizing Asynchronous Bounded-Delay Networks
IEEE Transactions on Communications

References

G. N. Frederickson and N. A. Lynch,
The impact of synchronous communication on the problem of electing a leader in a ring,
References

J. van Leeuwen, N. Santoro, J. Urrutia and S. Zaks,
Guessing Games and Distributed Computations in Synchronous Networks
Proceedings of the 14th International Colloquium on Automata, Languages and Programming (ICALP), Germany, July 1987, Springer-Verlag Lecture Notes in Computer Science No. 267, pp. 347-356.