The objective is to make the Lynch-Welch algorithm of Ch10 withstand any number of transient faults and at the same time up to $f$ Byzantine faults.
Algorithm 11 Lynch-Welch pulse synchronization algorithm, the code for a node $v \in V_g$. $S$ denotes a (to-be-determined) upper bound on $||\vec{p}_r||$ for each $r \in \mathbb{N}_{>0}$ and $T$ is the nominal round duration and it needs to be specified how Line 9 is implemented.

1: wait until $\text{getH()} = S$  \hfill // $H_w(0) \in [0, S)$ for all $w \in V$
2: for all round $r \in \mathbb{N}$ do
3: \hspace{1em} generate $r$-th pulse
4: \hspace{1em} $h \leftarrow \text{getH()}$
5: \hspace{1em} wait until $\text{getH()} = h + \vartheta S$  \hfill // all nodes are in round $r$
6: \hspace{1em} broadcast empty message to all nodes (including self)
7: \hspace{1em} wait until $\text{getH()} = h + (\vartheta^2 + \vartheta)S + \vartheta d$  \hfill // denote this time by $\tau_{v,r}$
8: \hspace{2em} // correct nodes’ messages should have arrived
9: \hspace{1em} for each node $w \in V$ do
10: \hspace{1.5em} compute $\Delta(w) \in [p_{w,r} - p_{v,r}, p_{w,r} - p_{v,r} + \delta]$  \hfill // denote $p_r := \max_{w \in V_g} \{p_{w,r}\}$
11: \hspace{2em} end for
12: $S \leftarrow \{\Delta(w) \mid w \in V\}$ (as multiset, i.e., values may repeat)
13: $\Delta \leftarrow \left( S_v^{(f+1)} + S_v^{(n-f)} \right) / 2$
14: wait until $\text{getH()} = h + \Delta + T$
15: end for
Initial requirements on round execution

Since we simulate an approximate agreement in a synchronous environment we need to make sure that the following holds:

1) Messages sent by correct nodes in a given round should be received by all correct nodes after they start the current round and before they compute the clock estimates, i.e., during $[p_{v,r}, \tau_{v,r}]$

2) $T$ is large enough to accommodate the adjustments for the next iteration, i.e., $H_v(\tau_{v,r}) \leq H_v(p_{v,r}) + \Delta + T$
4 nodes – 1 faulty and perturbation in values

\[ ||\bar{x}|| + 2\delta \]

\[ y_v = \frac{S_v^{f+1} + S_v^{n-f}}{2} \]

\[ ||\bar{y}|| \leq \frac{||\bar{x}||}{2} + \delta \]

\[ y_w = \frac{S_w^{f+1} + S_w^{n-f}}{2} \]
The objective is to make the Lynch-Welch algorithm of Ch10 withstand any number of transient faults and at the same time up to $f$ Byzantine faults.

- Main weaknesses of the original algorithm:
  1. assumed pretty strong synchronization
  2. assumed a small initial skew
  3. assumed receiving of at least $n-t$ values from correct nodes in each "window"
Algorithm 16 The loop of Algorithm 11, which is run alongside the local instances of the beat generation algorithm and Algorithm 17. Note that Algorithm 17 may reset the loop for stabilization purposes.

1: while true do
2:   generate pulse \hspace{1cm} // assume that $r \in \mathbb{N}$ is the pulse index
3:   $h \leftarrow \text{getH}()$
4:   wait until getH() = $h + \vartheta S$ \hspace{1cm} // all nodes are in round $r$
5:   broadcast empty message to all nodes (including self)
6:   wait until getH() = $h + (\vartheta^2 + \vartheta)S + \vartheta d$
   \hspace{1cm} // denote this time by $\tau_{v,r}$
   \hspace{1cm} // correct nodes’ messages should have arrived
7:   for each node $w \in V$ do
8:     compute $\Delta(w) \in [p_{w,r} - p_{v,r}, p_{w,r} - p_{v,r} + \delta]$ \hspace{1cm} // denote $p_r := \max_{w \in V \setminus \{v\}} \{p_{w,r}\}$
9:   end for
10:  $U \leftarrow \{\Delta(w) \mid w \in V\}$ (as multiset, i.e., values may repeat)
11:  $\Delta \leftarrow (U^{(f+1)} + U^{(n-f)}) / 2$
12:  wait until getH() = $h + \Delta + T$
13: end while

To prevent chaos the algorithm will be reset when events do not line up properly
Background Synchronization

We assume: some time after the number of existing faults falls below \( t \), the following can be achieved:

- Every correct node generates an event (\textbf{Beat}) at a "regular" period
- All events of non-faulty nodes in each wave of events are within some \( \sigma_h \) of each other
- We have lower and upper bounds on the period length between waves of events.

We will show how to obtain that in a later chapter.
Background Synchronization

We will make this state-machine self-stabilizing.
Can this replace LW?
Coordinating Two Independent Cycles

- To prevent chaos we need to coordinate the two streams of events.
- Our aim is to reduce the skew – waiting for the slow process to produce the desired beat increases the skew.
- We need another approach.
Beats and Feedbacks

Definition 13.2 (Feedback Mechanism). Nodes $v \in V_g$ generate beats at times $h_{v,i} \in \mathbb{R}$, $i \in \mathbb{N}$, such that for parameters $0 < B_1 < B_2 < B_3 \in \mathbb{R}$ and $\sigma_h$ (a skew bound) the following properties hold, for all $i \in \mathbb{N}$.

1. For all $v, w \in V_g$, we have that $|h_{v,i} - h_{w,i}| \leq \sigma_h$. 
**Definition 13.2** (Feedback Mechanism). Nodes $v \in V_g$ generate beats at times $h_{v,i} \in \mathbb{R}, i \in \mathbb{N}$, such that for parameters $0 < B_1 < B_2 < B_3 \in \mathbb{R}$ and $\sigma_h$ (a skew bound) the following properties hold, for all $i \in \mathbb{N}$.

1. For all $v, w \in V_g$, we have that $|h_{v,i} - h_{w,i}| \leq \sigma_h$.
2. If no $v \in V_g$ triggers its NEXT signal during $[\min_{w \in V_g} \{h_{w,i}\} + B_1, t]$ for some $t < \min_{w \in V_g} \{h_{w,i}\} + B_3$, then $\min_{w \in V_g} \{h_{w,i+1}\} > t$. 

The diagram illustrates the timing of beats and feedbacks with intervals $h$, $h + B_1$, $t$, and $h + B_3$, with no "next" signal indicated.
Definition 13.2 (Feedback Mechanism). Nodes $v \in V_g$ generate beats at times $h_{v,i} \in \mathbb{R}$, $i \in \mathbb{N}$, such that for parameters $0 < B_1 < B_2 < B_3 \in \mathbb{R}$ and $\sigma_h$ (a skew bound) the following properties hold, for all $i \in \mathbb{N}$.

1. For all $v, w \in V_g$, we have that $|h_{v,i} - h_{w,i}| \leq \sigma_h$.
2. If no $v \in V_g$ triggers its NEXT signal during $[\min_{w \in V_g} \{h_{w,i}\} + B_1, t]$ for some $t < \min_{w \in V_g} \{h_{w,i}\} + B_3$, then $\min_{w \in V_g} \{h_{w,i+1}\} > t$.
3. If all $v \in V_g$ trigger their NEXT signals during $[\min_{w \in V_g} \{h_{w,i}\} + B_2, t]$ for some $t \leq \min_{w \in V_g} \{h_{w,i}\} + B_3$, then $\max_{w \in V_g} \{h_{w,i+1}\} \leq t + \sigma_h$. 
Beats and Feedbacks

Definition 13.2 (Feedback Mechanism). Nodes $v \in V_g$ generate beats at times $h_{v,i} \in \mathbb{R}$, $i \in \mathbb{N}$, such that for parameters $0 < B_1 < B_2 < B_3 \in \mathbb{R}$ and $\sigma_h$ (a skew bound) the following properties hold, for all $i \in \mathbb{N}$.

1. For all $v, w \in V_g$, we have that $|h_{v,i} - h_{w,i}| \leq \sigma_h$.
2. If no $v \in V_g$ triggers its NEXT signal during $[\min_{w \in V_g} \{h_{w,i}\} + B_1, t]$ for some $t < \min_{w \in V_g} \{h_{w,i}\} + B_3$, then $\min_{w \in V_g} \{h_{w,i+1}\} > t$.
3. If all $v \in V_g$ trigger their NEXT signals during $[\min_{w \in V_g} \{h_{w,i}\} + B_2, t]$ for some $t \leq \min_{w \in V_g} \{h_{w,i}\} + B_3$, then $\max_{w \in V_g} \{h_{w,i+1}\} \leq t + \sigma_h$. 
Plan

• If events do not lineup – we will reset LW - an extreme measure that we take to converge from faults

• To obtain small skew we should not take such an action when events line up.

• To overcome the extra skew that the synchronization with the beats produce, we reduce the skew prior to that stage. We repeat the LW (approximate agreement) loop for several times in a row. (M iterations)

How to put all of that together?
Recall: Approximate Agreement Algorithm

The algorithm proceeds in rounds.

The Basic Iteration:

1. send $r_v$ to all.
2. receive $r_{w,v}$, the value sent by $w$ in this round. 
   \[
   \text{// replace any "missing" value by } r_v
   \]
3. $S_v := \{r_{w,v}\}$; \hspace{1cm} // ordered set
4. $o_v := (S_v^{(f+1)} + S_v^{(n-f)})/2$; \hspace{1cm} // the $(f+1)$st and $(n-f)$-th values in $S$
5. Return $o_v$

The initial range is reduced by half at the end of each iteration.

We will start with $S$ and end with $S(M)$ after $M$ iterations.
Beats and Pulses Alignment

If pulses are clustered we get the "no next" window followed by the "next" window.
The Meta Algorithm

Algorithm 17 Interface algorithm, actions for node $v \in V_g$ in response to a local event at time $t$. Runs in parallel to local instances of the beat generation algorithm and Algorithm 16.

1. $\triangleright$ The algorithm maintains local variable $i \in [M]$
2. if $v$ generates a pulse at time $t$ then
3. \hspace{1em} $i := i + 1 \mod M$
4. \hspace{1em} if $i = 0$ then
5. \hspace{2em} wait until local time $H_v(t) + \partial S(M)$
6. \hspace{2em} trigger NEXT signal
7. \hspace{1em} end if
8. end if

- Cycle for M rounds –
- Wait for the skew to pass
- Invoke a NEXT event
A well aligned **Beat** requires that the next "**Pulse 1**" will be within the "**green**" window.

Otherwise – invoke a **Reset**
Algorithm 17 Interface algorithm, actions for node $v \in V_g$ in response to a local event at time $t$. Runs in parallel to local instances of the beat generation algorithm and Algorithm 16.

1: $\quad \triangleright$ The algorithm maintains local variable $i \in [M]$
2: if $v$ generates a pulse at time $t$ then
3: $\quad i := i + 1 \mod M$
4: if $i = 0$ then
5: $\quad$ wait until local time $H_v(t) + \vartheta S(M)$
6: $\quad$ trigger NEXT signal
7: end if
8: end if
9: if $v$ generates a beat at time $t$ then
10: if $i \neq 0$ then $\quad \triangleright$ beats should align with every $M^{th}$ pulse, hence reset
11: $\quad$ reset($R^+$)
12: else if Algorithm 16 requires generating a pulse before $H_v(t) + R^-$ then
13: $\quad$ $\triangleright$ reset at pulse time $t'$ to avoid early pulse or message
14: $\quad$ reset($R^+ - (H_v(t') - H_v(t))$), where $t'$ is the current time
15: else if next pulse is not generated by local time $H_v(t) + R^+$ then
16: $\quad$ $\triangleright$ reset to avoid late pulse and
17: $\quad$ $\triangleright$ start listening for other nodes’ pulses on time
18: $\quad$ reset(0)
19: end if
20: end if
21: Function(reset($\tau$))
22: stop local instance of Algorithm 16
23: wait for $\tau$ local time
24: $i := 0$
25: initialize a new local instance of Algorithm 16
9: if \( v \) generates a beat at time \( t \) then
10: \hspace{1em} \textbf{if} \ i \neq 0 \textbf{then}
11: \hspace{2em} \triangleright \text{beats should align with every } M^{th} \text{ pulse, hence reset}
12: \hspace{2em} \text{reset}(R^+)

21: \textbf{Function}(\text{reset}(\tau))
22: \text{stop local instance of Algorithm 16}
23: \text{wait for } \tau \text{ local time}
24: i := 0
25: \text{initialize a new local instance of Algorithm 16}
9: if \( \nu \) generates a beat at time \( t \) then
10: \hspace{1em} if \( i \neq 0 \) then
\hspace{2em} \triangleright \text{beats should align with every } M^{th} \text{ pulse, hence reset}
11: \hspace{1em} \text{reset(} R^+ \text{)}
12: \hspace{1em} \text{else if Algorithm 16 requires generating a pulse before } H_\nu(t) + R^- \text{ then}
13: \hspace{1em} \triangleright \text{reset at pulse time } t' \text{ to avoid early pulse or message}
14: \hspace{1em} \text{reset(} R^+ - (H_\nu(t') - H_\nu(t)) \text{), where } t' \text{ is the current time}

21: Function(reset(\( \tau \)))
22: \text{stop local instance of Algorithm 16}
23: \text{wait for } \tau \text{ local time}
24: i := 0
25: \text{initialize a new local instance of Algorithm 16}
9: if \( v \) generates a beat at time \( t \) then

15: else if next pulse is not generated by local time \( H_v(t) + R^+ \) then

16: \( \triangleright \) reset to avoid late pulse and

17: \( \triangleright \) start listening for other nodes’ pulses on time

18: reset(0)

19: end if

20: end if

21: Function(reset(\( \tau \)))

22: stop local instance of Algorithm 16

23: wait for \( \tau \) local time

24: \( i := 0 \)

25: initialize a new local instance of Algorithm 16

We force an immediate pulse
9: if $v$ generates a beat at time $t$ then

15: else if next pulse is not generated by local time $H_v(t) + R^+$ then

16:   > reset to avoid late pulse and
17:   > start listening for other nodes’ pulses on time
18:   reset(0)
19: end if
20: end if $i=0$ and well aligned (green window)
21: Function(reset($\tau$))
22: stop local instance of Algorithm 16
23: wait for $\tau$ local time
24: $i := 0$
25: initialize a new local instance of Algorithm 16
9: if $v$ generates a beat at time $t$ then
10:   if $i \neq 0$ then
       \hspace{1em} \triangleright \text{beats should align with every } M^{th} \text{ pulse, hence reset}
11:       reset($R^+$)
12:   else if Algorithm 16 requires generating a pulse before $H_v(t) + R^-$ then
       \hspace{1em} \triangleright \text{reset at pulse time } t' \text{ to avoid early pulse or message}
13:       reset($R^+ - (H_v(t') - H_v(t))$), where $t'$ is the current time
14:   else if next pulse is not generated by local time $H_v(t) + R^+$ then
       \hspace{1em} \triangleright \text{reset to avoid late pulse and}
15:       \hspace{1em} \triangleright \text{start listening for other nodes' pulses on time}
16:       reset(0)
17:   end if
18: end if
19: i:=0 \text{ and well aligned (green window)}
20: end if
21: Function(reset($\tau$))
22: stop local instance of Algorithm 16
23: wait for $\tau$ local time
24: $i := 0$
25: initialize a new local instance of Algorithm 16

From the pseudocode given in Algorithm 17, it is straightforward to verify that $v \in V_{g}$ generates a pulse at a local time from $[H_v(h_{v,1}) + R^-, H_v(h_{v,1}) + R^+]$, and does not generate a pulse at a local time from $[H_v(h_{v,1}), H_v(h_{v,1}) + R^-]$. 
Definition 13.2 (Feedback Mechanism). Nodes \( v \in V_g \) generate beats at times \( h_{v,i} \in \mathbb{R}, \ i \in \mathbb{N} \), such that for parameters \( 0 < B_1 < B_2 < B_3 \in \mathbb{R} \) and \( \sigma_h \) (a skew bound) the following properties hold, for all \( i \in \mathbb{N} \).

1. For all \( v, w \in V_g \), we have that \( |h_{v,i} - h_{w,i}| \leq \sigma_h \).
2. If no \( v \in V_g \) triggers its NEXT signal during \( [\min_{w \in V_g} \{h_{w,i}\} + B_1, t] \) for some \( t < \min_{w \in V_g} \{h_{w,i}\} + B_3 \), then \( \min_{w \in V_g} \{h_{w,i+1}\} > t \).

Window without NEXT can be large
beat could be triggered w/o NEXT signals