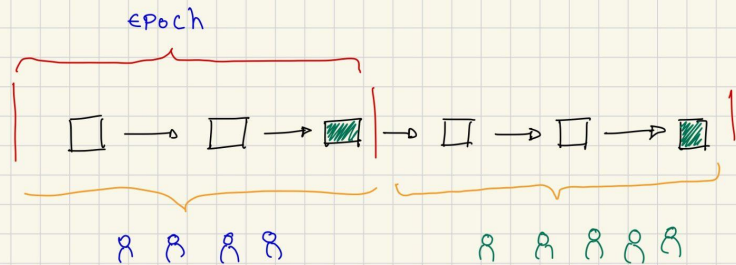



WARP SYNC



 → The last block that a given authority set finalized. This block should contain a digest signaling an authority set change from which we can fetch the next authority set

Warp Sync Proof

The warp sync proof is a message sent to the peer that contains accumulated proof of multiple authority set changes

$$M = \left(\underbrace{f_1 \dots f_n}_{\text{The vector of Warp Sync Fragments}}, c \right)$$

Indicate whether the warp sync has been completed

$$f_x = \left(\text{shaded square}, \underbrace{J^T, \text{stage}(B)} \right)$$

Justification for the header which proves it finality. In order to validate it the verifier must be aware of the authorities and set id for which the justification refers to.

Given the Warp Sync implementation part I, after the warp sync is completed a full state request is sent to a peer asking for the most up to date state, once valid gives to the node the most up to date state to operate.

Proposed changes to our current sync

The current worker poll is bounded to the **/sync** protocol. The worker pool execute the request depending on the strategy, and the strategy should be responsible to set the P2P protocol ID

We should rely on strategies, the strategies like **Warp Sync** or **Full Sync** does not have any knowledge about networking but has as dependency the current sync state and the grandpa state

The syncer starts a long term process, that runs forever until the end of the application and starts by calling the **strategy.NextAction()** which should return a set of tasks that should be sent to the workers

The workers executes the tasks and send the responses back to be handled by a strategy, that is actually implemented but only targeting the full sync approach

```
for {  
    TASKS := STRATEGY.NextActions()  
    WORKERS.Send(TASKS)  
  
    IF(STRATEGY.Finished()) {  
        STRATEGY = SyncService.DEFAULTSTRATEGY()  
    }  
}
```

Basically the presented approach pushes the responsibility to validate / import the responses to the strategies.

```

+     /// Handle a response from the remote to a warp proof request that we made.
+     ///
+     /// Returns next request.
+     pub fn on_warp_sync_data(
+         &mut self,
+         who: &PeerId,
+         response: warp::EncodedProof,
+     ) -> Result<OnWarpSyncData<B>, BadPeer> {
+         let import_result = if let Some(sync) = &mut self.warp_sync {
+             debug!(
+                 target: "sync",
+                 "Importing warp proof data from {}, {} bytes.",
+                 who,
+                 response.0.len(),
+             );
+             sync.import_warp_proof(response)
+         } else {
+             debug!(target: "sync", "Ignored obsolete warp sync response from {}");
+             return Err(BadPeer(who.clone(), rep::NOT_REQUESTED));
+         };
+
+         match import_result {
+             warp::WarpProofImportResult::StateRequest(request) =>
+                 Ok(OnWarpSyncData::StateRequest(who.clone(), request)),
+             warp::WarpProofImportResult::WarpProofRequest(request) =>
+                 Ok(OnWarpSyncData::WarpProofRequest(who.clone(), request)),
+             warp::WarpProofImportResult::BadResponse => {
+                 debug!(target: "sync", "Bad proof data received from {}");
+                 return Err(BadPeer(who.clone(), rep::BAD_BLOCK));
+             },
+         }
+     }
+ }

```

Warp Sync Strategy

Next Action():

Given the best block build the warp proof request

sender, receiver = request response channel

The sender goes into the task and the receiver stays here.

isFinalized():

listen on receiver:

```
STATE RESPONSE => IMPORT STATE ()  
UPDATE BABE EPOCHS ()  
RETURN TRUE
```

WARP RESPONSE =>

GRAND PA. validateAndImport(response)

IF(warp is finalized)

The next action should to request
the most up to date state

RETURN FALSE.