```
1  using Neo.Core;
2  using Neo.Cryptography;
3  using Neo.IO;
4  using Neo.Network;
5  using Neo.Network.Payloads;
6  using Neo.Plugins;
7  using Neo.SmartContract;
8  using Neo.Wallets;
9  using System;
10 using System.Collections.Generic;
11 using System.Linq;
12 using System.Threading;
13 using DbgViewTR;
14
15 namespace Neo.Consensus
16 {
17     public class ConsensusService : IDisposable
18     {
19         private ConsensusContext context = new ConsensusContext();
20         private LocalNode localNode; //network.LocalNode.cs
21         private Wallet wallet; //Neo.Wallets.Wallet.cs
22         private Timer timer;
23         private uint timer_height;
24         private byte timer_view;
25         private DateTime block_received_time;
26         private bool started = false;
27
28         public ConsensusService(LocalNode localNode, Wallet wallet)
29         {
30             TR.Enter();
31             this.localNode = localNode;
32             this.wallet = wallet;
33             this.timer = new Timer(OnTimeout, null, Timeout.Infinite,
                 Timeout.Infinite);
34             TR.Exit();
35         }
36
37         private bool AddTransaction(Transaction tx, bool verify)
38         {
39             TR.Enter();
40             if (Blockchain.Default.ContainsTransaction(tx.Hash) ||
41                 (verify && !tx.Verify(context.Transactions.Values)) ||
42                 !CheckPolicy(tx))
43             {
44                 Log($"reject tx: {tx.Hash}{Environment.NewLine}{tx.ToArray
                     ().ToHexString()}");
45                 RequestChangeView();
46                 return TR.Exit(false);
47             }
48             context.Transactions[tx.Hash] = tx;
49             if (context.TransactionHashes.Length == context.Transactions.Count)
50             {
51                 if (Blockchain.GetConsensusAddress(Blockchain.Default.GetValidators
                     (context.Transactions.Values).ToArray()).Equals
                     (context.NextConsensus))
52                 {
```

```
53                      Log($"send perpare response");
54                      context.State |= ConsensusState.SignatureSent;
55                      context.Signatures[context.MyIndex] = context.MakeHeader().Sign
                          (context.KeyPair);
56                      SignAndRelay(context.MakePrepareResponse(context.Signatures
                          [context.MyIndex]));
57                      CheckSignatures();
58                  }
59                  else
60                  {
61                      RequestChangeView();
62                      return TR.Exit(false);
63                  }
64              }
65              return TR.Exit(true);
66          }
67
68          private void Blockchain_PersistUnlocked(object sender, Block block)
69          {
70              TR.Enter();
71              Log($"persist block: {block.Hash}");
72              block_received_time = DateTime.Now;
73              InitializeConsensus(0);
74              TR.Exit();
75          }
76
77          private void CheckExpectedView(byte view_number)
78          {
79              TR.Enter();
80              if (context.ViewNumber == view_number) return;
81              if (context.ExpectedView.Count(p => p == view_number) >= context.M)
82              {
83                  InitializeConsensus(view_number);
84              }
85              TR.Exit();
86          }
87
88          private bool CheckPolicy(Transaction tx)
89          {
90              TR.Enter();
91              foreach (PolicyPlugin plugin in PolicyPlugin.Instances)
92                  if (!plugin.CheckPolicy(tx))
93                      return TR.Exit(false);
94              return TR.Exit(true);
95          }
96
97          private void CheckSignatures()
98          {
99              TR.Enter();
100             if (context.Signatures.Count(p => p != null) >= context.M &&
                  context.TransactionHashes.All(p => context.Transactions.ContainsKey(p)))
101             {
102                 Contract contract = Contract.CreateMultiSigContract(context.M,
                      context.Validators);
103                 Block block = context.MakeHeader();
104                 ContractParametersContext sc = new ContractParametersContext(block);
```

```
105                 for (int i = 0, j = 0; i < context.Validators.Length && j < context.M;
                        i++)
106                     if (context.Signatures[i] != null)
107                     {
108                         sc.AddSignature(contract, context.Validators[i],
                          context.Signatures[i]);
109                         j++;
110                     }
111                 sc.Verifiable.Scripts = sc.GetScripts();
112                 block.Transactions = context.TransactionHashes.Select(p =>
                      context.Transactions[p]).ToArray();
113                 Log($"relay block: {block.Hash}");
114                 if (!localNode.Relay(block))
115                     Log($"reject block: {block.Hash}");
116                 context.State |= ConsensusState.BlockSent;
117             }
118             TR.Exit();
119         }
120 
121         public void Dispose()
122         {
123             TR.Enter();
124             Log("OnStop");
125             if (timer != null) timer.Dispose();
126             if (started)
127             {
128                 Blockchain.PersistUnlocked -= Blockchain_PersistUnlocked;
129                 LocalNode.InventoryReceiving -= LocalNode_InventoryReceiving;
130                 LocalNode.InventoryReceived -= LocalNode_InventoryReceived;
131             }
132             TR.Exit();
133         }
134 
135         private void FillContext()
136         {
137             TR.Enter();
138             IEnumerable<Transaction> mem_pool = LocalNode.GetMemoryPool().Where(p =>
                  CheckPolicy(p));
139             foreach (PolicyPlugin plugin in PolicyPlugin.Instances)
140                 mem_pool = plugin.Filter(mem_pool);
141             List<Transaction> transactions = mem_pool.ToList();
142             Fixed8 amount_netfee = Block.CalculateNetFee(transactions);
143             TransactionOutput[] outputs = amount_netfee == Fixed8.Zero ? new
                  TransactionOutput[0] : new[] { new TransactionOutput
144             {
145                 AssetId = Blockchain.UtilityToken.Hash,
146                 Value = amount_netfee,
147                 ScriptHash = wallet.GetChangeAddress()
148             } };
149             while (true)
150             {
151                 ulong nonce = GetNonce();
152                 MinerTransaction tx = new MinerTransaction
153                 {
154                     Nonce = (uint)(nonce % (uint.MaxValue + 1ul)),
155                     Attributes = new TransactionAttribute[0],
```

```
156                     Inputs = new CoinReference[0],
157                     Outputs = outputs,
158                     Scripts = new Witness[0]
159                 };
160                 if (Blockchain.Default.GetTransaction(tx.Hash) == null)
161                 {
162                     context.Nonce = nonce;
163                     transactions.Insert(0, tx);
164                     break;
165                 }
166             }
167             context.TransactionHashes = transactions.Select(p => p.Hash).ToArray();
168             context.Transactions = transactions.ToDictionary(p => p.Hash);
169             context.NextConsensus = Blockchain.GetConsensusAddress
                  (Blockchain.Default.GetValidators(transactions).ToArray());
170             TR.Exit();
171         }
172 
173         private static ulong GetNonce()
174         {
175             TR.Enter();
176             byte[] nonce = new byte[sizeof(ulong)];
177             Random rand = new Random();
178             rand.NextBytes(nonce);
179             return TR.Exit(nonce.ToUInt64(0));
180         }
181 
182         private void InitializeConsensus(byte view_number)
183         {
184             TR.Enter();
185             lock (context)
186             {
187                 if (view_number == 0)
188                     context.Reset(wallet);
189                 else
190                     context.ChangeView(view_number);
191                 if (context.MyIndex < 0) return;
192                 Log($"initialize: height={context.BlockIndex} view={view_number}
                    index={context.MyIndex} role={(context.MyIndex ==
                    context.PrimaryIndex ? ConsensusState.Primary :
                    ConsensusState.Backup)}");
193                 if (context.MyIndex == context.PrimaryIndex)
194                 {
195                     context.State |= ConsensusState.Primary;
196                     if (!context.State.HasFlag(ConsensusState.SignatureSent))
197                     {
198                         FillContext();
199                     }
200                     if (context.TransactionHashes.Length > 1)
201                     {
202                         InvPayload invPayload = InvPayload.Create(InventoryType.TX,
                      context.TransactionHashes.Skip(1).ToArray());
203                         foreach (RemoteNode node in localNode.GetRemoteNodes())
204                             node.EnqueueMessage("inv", invPayload);
205                     }
206                     timer_height = context.BlockIndex;
```

```
207                     timer_view = view_number;
208                     TimeSpan span = DateTime.Now - block_received_time;
209                     if (span >= Blockchain.TimePerBlock)
210                         timer.Change(0, Timeout.Infinite);
211                     else
212                         timer.Change(Blockchain.TimePerBlock - span,
                          Timeout.InfiniteTimeSpan);
213                 }
214                 else
215                 {
216                     context.State = ConsensusState.Backup;
217                     timer_height = context.BlockIndex;
218                     timer_view = view_number;
219                     timer.Change(TimeSpan.FromSeconds(Blockchain.SecondsPerBlock <<
                          (view_number + 1)), Timeout.InfiniteTimeSpan);
220                 }
221             }
222             TR.Exit();
223         }
224 
225         private void LocalNode_InventoryReceived(object sender, IInventory inventory)
226         {
227             TR.Enter();
228             ConsensusPayload payload = inventory as ConsensusPayload;
229             if (payload != null)
230             {
231                 lock (context)
232                 {
233                     if (payload.ValidatorIndex == context.MyIndex) { TR.Exit();
                          return; }
234 
235                     if (payload.Version != ConsensusContext.Version)
236                     {
237                         TR.Exit();
238                         return;
239                     }
240                     if (payload.PrevHash != context.PrevHash || payload.BlockIndex !=
                          context.BlockIndex)
241                     {
242                         // Request blocks
243 
244                         if (Blockchain.Default?.Height + 1 < payload.BlockIndex)
245                         {
246                             Log($"chain sync: expected={payload.BlockIndex} current:
                              {Blockchain.Default?.Height} nodes=
                              {localNode.RemoteNodeCount}");
247 
248                             localNode.RequestGetBlocks();
249                         }
250                         TR.Exit();
251                         return;
252                     }
253 
254                     if (payload.ValidatorIndex >= context.Validators.Length) { TR.Exit
                          (); return; }
255                     ConsensusMessage message;
```

```
256                     try
257                     {
258                         message = ConsensusMessage.DeserializeFrom(payload.Data);
259                     }
260                     catch
261                     {
262                         TR.Exit();
263                         return;
264                     }
265                     if (message.ViewNumber != context.ViewNumber && message.Type !=
                          ConsensusMessageType.ChangeView)
266                     {
267                         TR.Exit();
268                         return;
269                     }
270                     switch (message.Type)
271                     {
272                         case ConsensusMessageType.ChangeView:
273                             OnChangeViewReceived(payload, (ChangeView)message);
274                             break;
275                         case ConsensusMessageType.PrepareRequest:
276                             OnPrepareRequestReceived(payload, (PrepareRequest)
                              message);
277                             break;
278                         case ConsensusMessageType.PrepareResponse:
279                             OnPrepareResponseReceived(payload, (PrepareResponse)
                              message);
280                             break;
281                     }
282                 }
283             }
284             TR.Exit();
285         }
286 
287         private void LocalNode_InventoryReceiving(object sender,
              InventoryReceivingEventArgs e)
288         {
289             TR.Enter();
290             Transaction tx = e.Inventory as Transaction;
291             if (tx != null)
292             {
293                 lock (context)
294                 {
295                     if (!context.State.HasFlag(ConsensusState.Backup) || !
                          context.State.HasFlag(ConsensusState.RequestReceived) ||
                          context.State.HasFlag(ConsensusState.SignatureSent) ||
                          context.State.HasFlag(ConsensusState.ViewChanging))
296                         return;
297                     if (context.Transactions.ContainsKey(tx.Hash)) return;
298                     if (!context.TransactionHashes.Contains(tx.Hash)) return;
299                     AddTransaction(tx, true);
300                     e.Cancel = true;
301                 }
302             }
303             TR.Exit();
304         }
```

```
305
306         protected virtual void Log(string message)
307         {
308             // something should be here.
309             TR.Enter();
310             TR.Exit();
311         }
312
313         private void OnChangeViewReceived(ConsensusPayload payload, ChangeView
              message)
314         {
315             TR.Enter();
316             Log($"{nameof(OnChangeViewReceived)}: height={payload.BlockIndex} view=
                  {message.ViewNumber} index={payload.ValidatorIndex} nv=
                  {message.NewViewNumber}");
317             if (message.NewViewNumber <= context.ExpectedView[payload.ValidatorIndex])
318             {
319                 TR.Exit();
320                 return;
321             }
322             context.ExpectedView[payload.ValidatorIndex] = message.NewViewNumber;
323             CheckExpectedView(message.NewViewNumber);
324             TR.Exit();
325         }
326
327         private void OnPrepareRequestReceived(ConsensusPayload payload, PrepareRequest
              message)
328         {
329             TR.Enter();
330             Log($"{nameof(OnPrepareRequestReceived)}: height={payload.BlockIndex}
                  view={message.ViewNumber} index={payload.ValidatorIndex} tx=
                  {message.TransactionHashes.Length}");
331             if (!context.State.HasFlag(ConsensusState.Backup) || context.State.HasFlag
                  (ConsensusState.RequestReceived))
332             {
333                 TR.Exit();
334                 return;
335             }
336             if (payload.ValidatorIndex != context.PrimaryIndex) return;
337             if (payload.Timestamp <= Blockchain.Default.GetHeader
                  (context.PrevHash).Timestamp || payload.Timestamp >
                  DateTime.Now.AddMinutes(10).ToTimestamp())
338             {
339                 Log($"Timestamp incorrect: {payload.Timestamp}");
340                 TR.Exit();
341                 return;
342             }
343             context.State |= ConsensusState.RequestReceived;
344             context.Timestamp = payload.Timestamp;
345             context.Nonce = message.Nonce;
346             context.NextConsensus = message.NextConsensus;
347             context.TransactionHashes = message.TransactionHashes;
348             context.Transactions = new Dictionary<UInt256, Transaction>();
349             if (!Crypto.Default.VerifySignature(context.MakeHeader().GetHashData(),
                  message.Signature, context.Validators
                  [payload.ValidatorIndex].EncodePoint(false))) { TR.Exit(); return; }
```

```
350             context.Signatures = new byte[context.Validators.Length][];
351             context.Signatures[payload.ValidatorIndex] = message.Signature;
352             Dictionary<UInt256, Transaction> mempool = LocalNode.GetMemoryPool
                  ().ToDictionary(p => p.Hash);
353             foreach (UInt256 hash in context.TransactionHashes.Skip(1))
354             {
355                 if (mempool.TryGetValue(hash, out Transaction tx))
356                     if (!AddTransaction(tx, false))
357                     {
358                         TR.Exit();
359                         return;
360                     }
361             }
362             if (!AddTransaction(message.MinerTransaction, true)) { TR.Exit();
                  return; }
363             if (context.Transactions.Count < context.TransactionHashes.Length)
364             {
365                 UInt256[] hashes = context.TransactionHashes.Where(i => !
                      context.Transactions.ContainsKey(i)).ToArray();
366                 LocalNode.AllowHashes(hashes);
367                 InvPayload msg = InvPayload.Create(InventoryType.TX, hashes);
368                 foreach (RemoteNode node in localNode.GetRemoteNodes())
369                     node.EnqueueMessage("getdata", msg);
370             }
371             TR.Exit();
372         }
373
374         private void OnPrepareResponseReceived(ConsensusPayload payload,
              PrepareResponse message)
375         {
376             TR.Enter();
377             Log($"{nameof(OnPrepareResponseReceived)}: height={payload.BlockIndex}
                  view={message.ViewNumber} index={payload.ValidatorIndex}");
378             if (context.State.HasFlag(ConsensusState.BlockSent)) { TR.Exit();
                  return; }
379             if (context.Signatures[payload.ValidatorIndex] != null) { TR.Exit();
                  return; }
380             Block header = context.MakeHeader();
381             if (header == null || !Crypto.Default.VerifySignature(header.GetHashData
                  (), message.Signature, context.Validators
                  [payload.ValidatorIndex].EncodePoint(false))) { TR.Exit(); return; }
382             context.Signatures[payload.ValidatorIndex] = message.Signature;
383             CheckSignatures();
384             TR.Exit();
385         }
386
387         private void OnTimeout(object state)
388         {
389             TR.Enter();
390             lock (context)
391             {
392                 if (timer_height != context.BlockIndex || timer_view !=
                      context.ViewNumber) { TR.Exit(); return; }
393                 Log($"timeout: height={timer_height} view={timer_view} state=
                      {context.State}");
394                 if (context.State.HasFlag(ConsensusState.Primary) && !
```

```
                    context.State.HasFlag(ConsensusState.RequestSent))
395                 {
396                     Log($"send perpare request: height={timer_height} view=
                        {timer_view}");
397                     context.State |= ConsensusState.RequestSent;
398                     if (!context.State.HasFlag(ConsensusState.SignatureSent))
399                     {
400                         context.Timestamp = Math.Max(DateTime.Now.ToTimestamp(),
                        Blockchain.Default.GetHeader(context.PrevHash).Timestamp + 1);
401                         context.Signatures[context.MyIndex] = context.MakeHeader
                        ().Sign(context.KeyPair);
402                     }
403                     SignAndRelay(context.MakePrepareRequest());
404                     timer.Change(TimeSpan.FromSeconds(Blockchain.SecondsPerBlock <<
                        (timer_view + 1)), Timeout.InfiniteTimeSpan);
405                 }
406                 else if ((context.State.HasFlag(ConsensusState.Primary) &&
                    context.State.HasFlag(ConsensusState.RequestSent)) ||
                    context.State.HasFlag(ConsensusState.Backup))
407                 {
408                     RequestChangeView();
409                 }
410             }
411             TR.Exit();
412         }
413
414         private void RequestChangeView()
415         {
416             TR.Enter();
417             context.State |= ConsensusState.ViewChanging;
418             context.ExpectedView[context.MyIndex]++;
419             Log($"request change view: height={context.BlockIndex} view=
                {context.ViewNumber} nv={context.ExpectedView[context.MyIndex]} state=
                {context.State}");
420             timer.Change(TimeSpan.FromSeconds(Blockchain.SecondsPerBlock <<
                (context.ExpectedView[context.MyIndex] + 1)), Timeout.InfiniteTimeSpan);
421             SignAndRelay(context.MakeChangeView());
422             CheckExpectedView(context.ExpectedView[context.MyIndex]);
423             TR.Exit();
424         }
425
426         private void SignAndRelay(ConsensusPayload payload)
427         {
428             TR.Enter();
429             ContractParametersContext sc;
430             try
431             {
432                 sc = new ContractParametersContext(payload);
433                 wallet.Sign(sc);
434             }
435             catch (InvalidOperationException)
436             {
437                 TR.Exit();
438                 return;
439             }
440             sc.Verifiable.Scripts = sc.GetScripts();
```

```
441                 localNode.RelayDirectly(payload);
442                 TR.Exit();
443             }
444 
445             public void Start()
446             {
447                 TR.Enter();
448                 Log("OnStart");
449                 started = true;
450                 Blockchain.PersistUnlocked += Blockchain_PersistUnlocked;
451                 LocalNode.InventoryReceiving += LocalNode_InventoryReceiving;
452                 LocalNode.InventoryReceived += LocalNode_InventoryReceived;
453                 InitializeConsensus(0);
454                 TR.Exit();
455             }
456         }
457     }
458 
```