Update_3.0
Consensus
* (This update is regarding consensus implementation)
Low : 4
Medium : 1
High : 0

Priority : Low
Issue :
https://github.com/quras-official/quras-blockchain-csharp/blob/master/QurasCore/Consensus/ConsensusContext.cs#L43
ExpectedView[MyIndex] should be set to proper view_number.

Priority : Low
Issue :
https://github.com/quras-official/quras-blockchain-csharp/blob/7d30972c8251547541554796bf85a606bc3f4c09/QurasCore/Consensus/ConsensusService.cs#L292

```
private void LocalNode_InventoryReceived(object sender, IInventory inventory)
{
    ConsensusPayload payload = inventory as ConsensusPayload;
    if (payload != null)
    {
        lock (context)
        {
            if (payload.ValidatorIndex == context.MyIndex) return;
            if (payload.Version != ConsensusContext.Version || payload.PrevHash != context.PrevHash || payload.BlockIndex != context.BlockIndex)
                return;
            if (payload.ValidatorIndex >= context.Validators.Length) return;
            ConsensusMessage message;
            try
            {
                message = ConsensusMessage.DeserializeFrom(payload.Data);
            }
            catch (FormatException)
            {
                return;
            }
```

Other errors are ignored, either catch should be generic or proper catch for each error should be handled otherwise function executes even if it encountered error. This can lead to incorrect message data in localnode receive.

Priority : Medium
Issue :
https://github.com/quras-official/quras-blockchain-csharp/blob/7d30972c8251547541554796bf85a606bc3f4c09/QurasCore/Consensus/ConsensusService.cs#L276

The LocalNode_InventoryReceived function should only fail directly in case the versions of payload don't match, in case the prev hash is now equal to context hash new blocks should be requested.

```
private void LocalNode_InventoryReceived(object sender, IInventory inventory)
{
    ConsensusPayload payload = inventory as ConsensusPayload;
    if (payload != null)
    {
        lock (context)
        {
            if (payload.ValidatorIndex == context.MyIndex) return;
            if (payload.Version != ConsensusContext.Version || payload.PrevHash != context.PrevHash || payload.BlockIndex != context.BlockIndex)
                return;
```

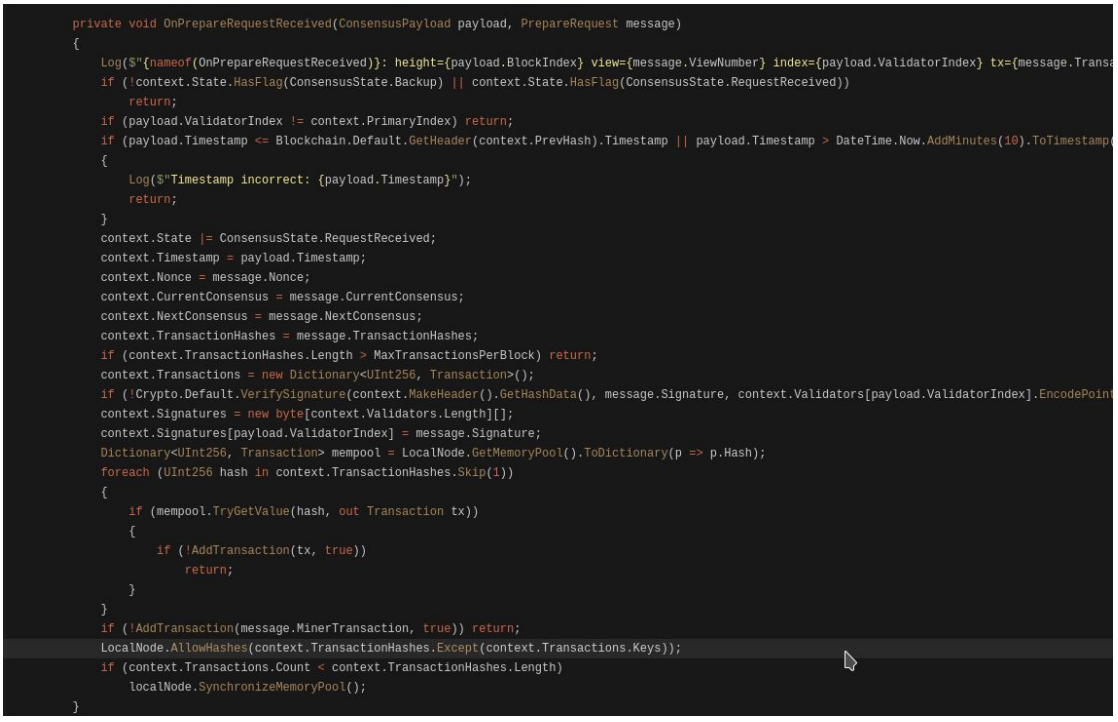
A rough flow will look like this :

```
if (payload.Version != ConsensusContext.Version)
                return;
```

if (payload.PrevHash != context.PrevHash || payload.BlockIndex != context.BlockIndex)
.... get blocks from localnode

Priority : Low
Issue : https://github.com/quras-official/quras-blockchain-csharp/blob/7d30972c8251547541554796bf85a606bc3f4c09/QurasCore/Consensus/ConsensusService.cs#L344

It is not necessary to call “localnode synchronize memory pool” and allow hash should not be called every time (because of unnecessary lock and enqueue happens inside the synchronize pool) This will lead to unnecessary memory utilization.



```
private void OnPrepareRequestReceived(ConsensusPayload payload, PrepareRequest message)
{
    Log($"{nameof(OnPrepareRequestReceived)}: height={payload.BlockIndex} view={message.ViewNumber} index={payload.ValidatorIndex} tx={message.Transa
    if (!context.State.HasFlag(ConsensusState.Backup) || context.State.HasFlag(ConsensusState.RequestReceived))
        return;
    if (payload.ValidatorIndex != context.PrimaryIndex) return;
    if (payload.Timestamp <= Blockchain.Default.GetHeader(context.PrevHash).Timestamp || payload.Timestamp > DateTime.Now.AddMinutes(10).ToTimestamp(
    {
        Log($"Timestamp incorrect: {payload.Timestamp}");
        return;
    }
    context.State |= ConsensusState.RequestReceived;
    context.Timestamp = payload.Timestamp;
    context.Nonce = message.Nonce;
    context.CurrentConsensus = message.CurrentConsensus;
    context.NextConsensus = message.NextConsensus;
    context.TransactionHashes = message.TransactionHashes;
    if (context.TransactionHashes.Length > MaxTransactionsPerBlock) return;
    context.Transactions = new Dictionary<UInt256, Transaction>();
    if (!Crypto.Default.VerifySignature(context.MakeHeader().GetHashData(), message.Signature, context.Validators[payload.ValidatorIndex].EncodePoint
    context.Signatures = new byte[context.Validators.Length][];
    context.Signatures[payload.ValidatorIndex] = message.Signature;
    Dictionary<UInt256, Transaction> mempool = LocalNode.GetMemoryPool().ToDictionary(p => p.Hash);
    foreach (UInt256 hash in context.TransactionHashes.Skip(1))
    {
        if (mempool.TryGetValue(hash, out Transaction tx))
        {
            if (!AddTransaction(tx, true))
                return;
        }
    }
    if (!AddTransaction(message.MinerTransaction, true)) return;
    LocalNode.AllowHashes(context.TransactionHashes.Except(context.Transactions.Keys));
    if (context.Transactions.Count < context.TransactionHashes.Length)
        localNode.SynchronizeMemoryPool();
}
```

inside of the if statement (context.Transactions.Count < context.TransactionHashes.Length) hashes should be calculated from context TransactionHashes and node enqueueMessage for "getData" should be called directly.

Priority : Low
Issue :
https://github.com/quras-official/quras-blockchain-csharp/blob/55d6151ca7924407c60b13641a714b4980be3775/QurasCore/Network/RemoteNode.cs#L129

EnqueueMessage Should only happen if the hash length is greater then zero and not for all. This will lead to in correct enqueuing of the “inv” messages.