Problem 1:

Two suggested protocols to increase transaction rate to 70/second are:

1. Increase size of block from 1MB to 10MB.
2. Decrease the block interval time to 1 min (from 10 minute).

Issues with increasing block size: This would require a lot more storage, and would make the full nodes more expensive to operate. And eventaully fully operating nodes will decrease, and the system will become more centralized. There might be more double spending attacks due to slower propogation speeds. Achieving consensus would be difficult since validating a block would require lot more efforts.

Issues with decreasing block interval time: Propogation of the block in the network, takes some time. If we reduce the block interval time, block might not be propogated fully in the network till that time. We might not be able to achieve consensus in due time. Block propogation and latency would also lead to more orphan nodes, since they might be delayed in propogating. It would also have an envionmental effect, since power would be used more since block rate is high.

Problem 2:

```
1 #include <openssl/conf.h>
2 #include <openssl/evp.h>
3 #include <openssl/err.h>
4 #include <openssl/pem.h>
5 #include <openssl/crypto.h>
6 #include <openssl/sha.h>
7 #include <openssl/rsa.h>
8
9 #include <bits/stdc++.h>
10
11 // bits denote the modulus size
12 const int bits = 2048;
13 // string needed for conversion to base58
14 static const char* pszBase58 = "123456789ABCDEFGHJKLMNPQRSTUVWXYZ"
15                                 +"abcdefghijkmnopqrstuvwxyz";
16 // Our RSA object to store key
17 RSA *rsa = NULL;
18 BIGNUM *bne = NULL;
19 // Key variables, that would store data in raw format
20 EVP_PKEY *privateKey, *publicKey;
21 // public key in DER format
22 uint8_t* pubKeyDER;
23 // length of DER public key
24 int key_len;
25
26 void set_up() {
27     // standard procedure taken from assignment1 crypto.cpp
28     bne = BN_new();
29     int ret = BN_set_word(bne, RSA_F4);
30     assert(ret==1);
31
32     // generate a new object from the constructor functions
33     rsa = RSA_new();
34     privateKey = EVP_PKEY_new();
35     publicKey = EVP_PKEY_new();
36 }
37
```

```
38 void gen_keys() {
39     // generates a key pair and stores it in the RSA structure provided in rsa.
40     int ret=RSA_generate_key_ex(rsa,bits,bne,NULL);
41     assert(ret==1);
42 
43     // encode and decode a public key.
44     key_len = i2d_RSA_PUBKEY(rsa, &pubKeyDER);
45     // store rsa value generated in keys.
46     EVP_PKEY_assign_RSA(privateKey, rsa);
47     EVP_PKEY_assign_RSA(publicKey, rsa);
48 }
49 
50 void print_keys() {
51     // print keys to standard buffer.
52     PEM_write_PrivateKey(stdout, privateKey, NULL, NULL, 0, NULL, NULL);
53     PEM_write_PUBKEY(stdout, publicKey);
54 }
55 
56 void free_structures() {
57     BN_free(bne); // free the big number structure
58     free(pubKeyDER); // free DER formatted key
59     RSA_free(rsa);
60 }
61 
62 /*
63 CITE:
64     Following implementation is taken from: bitcoin github repo
65     link: https://github.com/bitcoin/bitcoin/blob/master/src/base58.cpp
66 */
67 std::string EncodeBase58(const unsigned char* pbegin, const unsigned char* pend)
68 {
69     // Skip & count leading zeroes.
70     int zeroes = 0;
71     int length = 0;
72     while (pbegin != pend && *pbegin == 0) {
73         pbegin++;
74         zeroes++;
75     }
76     // Allocate enough space in big-endian base58 representation.
77     int size = (pend - pbegin) * 138 / 100 + 1; // log(256) / log(58), rounded up.
78     std::vector<unsigned char> b58(size);
79     // Process the bytes.
80     while (pbegin != pend) {
81         int carry = *pbegin;
82         int i = 0;
83         // Apply "b58 = b58 * 256 + ch".
84         for (std::vector<unsigned char>::reverse_iterator it = b58.rbegin();
85                 (carry != 0 || i < length) && (it != b58.rend()); it++, i++) {
86             carry += 256 * (*it);
87             *it = carry % 58;
88             carry /= 58;
89         }
90 
91         assert(carry == 0);
92         length = i;
93         pbegin++;
94     }
```

```
95      // Skip leading zeroes in base58 result.
96      std::vector<unsigned char>::iterator it = b58.begin() + (size - length);
97      while (it != b58.end() && *it == 0)
98          it++;
99      // Translate the result into a string.
100     std::string str;
101     str.reserve(zeroes + (b58.end() - it));
102     str.assign(zeroes, '1');
103     while (it != b58.end())
104         str += pszBase58[*(it++)];
105     return str;
106 }
107 // cite end..
108
109 int main() {
110     set_up();
111     gen_keys();
112     print_keys();
113
114     // compute hash of PEM formatted public key.
115     // This procedure is taken from Lec 3 slides.
116     unsigned char h1[SHA256_DIGEST_LENGTH];
117     SHA256_CTX sha256;
118     SHA256_Init(&sha256);
119     SHA256_Update(&sha256, &pubKeyDER[0], key_len);
120     SHA256_Final(h1, &sha256);
121
122     // Encode it in base58 using the standard implementation.
123     auto bitcoin_addr = EncodeBase58(h1, h1+SHA256_DIGEST_LENGTH);
124     // print the bitcoint address generated.
125     std::cout << bitcoin_addr << "\n";
126
127     free_structures();
128 }
```

Public key generated in this way is:

```
1 -----BEGIN PUBLIC KEY-----
2 MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAuqdKYAffbvISUIzFlMQY
3 FrJP/oyaTF8K+t8dynVKXR+0wgby8D68l6+G2fa97lp3gmZ3mfT8NjNskXP5CTL9
4 tP/x3XkGxYz5fVoByZNa160Udscghe2A2qDtTxJMdRVTDKU5M7UduQxK5iUkjVpi
5 Ejphrksw7qBLGrP4KCDbunnkZdAE/ns3LESi5SzwZ/aWuutZLJxaAiZaNTOHc9NQ
6 hXeOkyKC8H2+XIxJFUZsfExNTmhpls4+EE2KYD7Hqk9J4VYE1N6vMCvCC45H7A/B
7 X4vRtx5kGAv8anJdXH+rRChyQr5RakIQz4fDOhEw8hcIgTx6QXbEOtrxtTlWi3H/
8 1QIDAQAB
9 -----END PUBLIC KEY-----
```

The bitcoin address generated for this key is:

```
1     addr := Cgwj8VL9QLhGg9RG2oHoiVsSPqD8VdUze2coUDgxQWf3
```

Problem 3:

In bitcoin blockchain we can have double spending attacks. Let's see how:

Suppose Alice bought cocaine from Bob by paying him in bitcoin. Seeing this transaction in the most recent block Bob might think that transaction is successful, and would give alway cocaine to Alice. Now if the next random node this is selected in the next round happens to be controlled by Alice. So she might ignore the block including her cocaine transaction, and could build on the second most recent block, not including the cocaine transaction. So next time any miner would see 2 branches of equal length, being

honest it would the would be equally probable to build new block on either of them. Since There is no way to distinguish that one of the block tries to double spend, there are approximately 50% chances that double spend would occur.

51$\%$ attack: If there is a node that has > 51% hash power, then it can double by simply growing over the other transaction block, to a large extent, such that even 6 cofirmations won't help. So in above example if Bob was satisfied after seeing 6 more blocks on top of current transaction's block, Alice might start a new chain, and since she has hash power > 51%, she can easily take over the other chain.

Problem 4:

(a)

Since the miner is ahead of public blockchain by two secret blocks, all the mining efforts of the rest of the network will be wasted. Other miners would mine on top of what they think is the longest chain, so after the selfish miner announces, that branch would instantly become the new longest chain. Eventaully the rest of block (found by other miners), would become orphan. So in nutshell, gain in selfish mining is that effective share of mining rewards would increase.

(b)

First of all using time stamp is not a good way since we would encounter network latency, and it would not be synchronous over the blockchain network. But ignoring them, we can get an idea of selfish miners, but we can never be 100% sure that the current miner, who's timestamp are close is a selfish miner. Since there might be a miner who was lucky enough to find the next block early. So we cannot define a hard and fast rule to detect selfish miner.

Problem 5:

Problem 6:

Assuming that other miners have detected misbehaving miner's block, the next randomly selected node can "boycott", by not building on top of the misbehaving miner's block. In other words they can boycott a particular address correspoding to the coinbase transaction of this block.

Since there is no real identity in bitcoin blockchain, we cannot really identify the misbehaving node, based on this public key/wallet address. Since detecting misbehaving node is hard, the misbehaving node can simply change his public key, and can again misbehave. He wouldn't be affected much in this case, so "boycott" might not prevent him from misbehaving.

Problem 7:

As a blockchain designer, we can change the puzzle from "find a block whose hash is below certain value" to "find a block for which the hash of a signature on the block is below certain target". So in this case pool manager would have to share his private key with all the pool members, and this would be risky since members might steal money from his wallet. Other alternative would be that pool manager does the signing work and the members compute hash values. But since signing computationally more expensive than computing hash value, this scheme would not work either. And we can prevent pool mining.

Problem 8:

```
1 pragma solidity ^0.5.0;
2
3 // Our contract name
4 contract MyFirstContract {
5   // denotes account balance as uint
6   uint public balance;
```

```
7
8   // set balance to be 100 (initial value)
9   constructor() public {
10    balance = 100;
11  }
12
13  // returns the current value of balance paramter of our contract
14  function getBalance() public returns(uint) {
15    return balance;
16  }
17 }
```

MyFirstContract.sol

```
1 // method to request a usable contract abstraction for a specific Solidity contract
2 var MyFirstContract = artifacts.require("./MyFirstContract.sol");
3
4 // Include this in exports.
5 module.exports = function(deployer) {
6   // deploy this contract on Ethereum Network
7   deployer.deploy(MyFirstContract);
8 };
```

2_deploy_contracts.js

```
1 // Export the development configs.
2 module.exports = {
3     networks: {
4         development: {
5         host: "localhost",     // local ethureum network
6         port: 8545,            // Port of operation for ganache
7         network_id: "*"        // * to match any network ID, it is a required field
8         }
9     }
10 }
```

truffle-config.js

Problem 9:

Pros of Ethereum over Bitcoin are:

1. Ethereum provide us with EVM, which allow code to be verified and executed on the blockchain. This would provide us guarantee that it will be run the same way on everyone's machine.
2. There is no limit on block size, hence miners don't have to wait for block to fill, or remove some transaction in order to make block in size limit.
3. Ethereum Platform provides people to run local instances for personal use, blockchain does not have such a feature.
4. Block interval time in Ethereum is far less(10 sec) than compared to that of bitcoin. On an average there are about 25 transactions per second. Creating it more active form of currency.
5. Ethereum provides us smart contract directly, without any requirement of tweaking in the case of Bitcoin.

Cons of Ethereum over Bitcoin:

1. Number of bitcoin is fixed to 21 million, while number of Ethereum is around 91 million, Having large number of coin in some sense means that value would be low. So Bitcoin can be a better option in terms of value.

2. Bitcoin scripts are a bit restrictive(CFG based), so that is good for securing purpose.
3. It is assumed that in Bitcoin, it is really different to find if 2 address are linked (belong to same owner etc). So this provides a higher degree of privacy.
4. There does not exist instances of bitcoin blockchain, as in the case of Ethereum. This would somehow prevent the value of Bitcoin blockchain from falling.
5. Being first of its kind Bitcoin blockchain, has been the most popular among its peers. So it is more beneficial to use bitcoin.

But we shouldn't really try to compare Ethereum and Bitcoin on the same scale, as they are targeted for different applications.