WindfarmNetwork

WindfarmNetwork é um simulador da rede de um parque eólico. Uma windfarm (como normalmente são conhecidos) é composto por multiplas turbinas (clientes) e uma central de monitoramento (servidor). Durante o dia a dia dessas turbinas, é necessário uma comunicação segura e rápida com o monitoramento, afim de repassar 'status' de cada um dos sensores presentes nos mais de 1000 componentes da turbina, e notificar a central sobre eventuais falhas. Com isso, automaticamente a central consegue auxiliar o controlador da turbina sobre como reagir a cenário pré-definidos. Mas há também os casos, quando a turbina excede o numero de alarmes (um threshold) que é conhecido como 'tripagem', onde a turbina precisa de uma manutenção in-loco.

O objetivo desse pequeno projeto é simular como seria um protocolo de comunicação simples entre essas turbinas e seu respectivo monitor. Não foram adicionados 'sensores' nessas turbinas virtuais, porém, para compensar, foram descritas algumas funções (vento, geração de energia, fator de transformação do vento, etc) que visam tornar o projeto um "toy-model" da realidade. Além disso, como não é possível simular manutenções reais nesses equipamentos, para cada falha ou delay de comunicação, foi posto de timer que visa simular situações adversas de turbinas fora de comunicação ou que demoram a responder.

Especificação da comunicação

Para desenvolver o projeto, utilizamos da arquitetura de cliente-servidor, no qual as turbinas estão representando diversos clientes, que a todo momento estão trocando informações com o servidor (que é a central de monitoramento). A ideia é que tenhamos um servidor multithread, afim de em paralelo lidar com multiplas requisições e decisões sobre a performance das turbinas. Para o caso de testes, forma utilizados no máximo 3 turbinas simultaneamente, mas nada impede de expandir essa arquitetura para lidar com simulações maiores (+100 turbinas)

O programa foi separado em 2 arquivos: 'turbina.py' e 'monitoramento.py'. Este primeiro, guarda todas as informações da turbina, como e.g geração do vento, fator de transformação do vento (o quanto está transformando o vento em energia mecanica) e até mesmo o numero de falhas que a turbina possui atualmente. Enquanto isso, no monitoramento, temos guardado toda a lógica de como essas turbinas devem se comportar conforme determinados padrões de mensagens: Se uma turbina estiver com o vento acima de 19m/s, ela deve ser desligada e retornar após 10s de espera. Isso faz

com que simulemos ações básicas que um monitoramento real faz com turbinas no dia a dia.

Mas específicamente, foi-se posto uma abordagem em que, tirando as tomadas de decisões, toda a lógica seria pertencente a turbina: controle de falhas, mudanças de potencia de maquina, etc. Todos os casos citados são pequenas variáveis que a propria turbina ou o monitoramento podem ter controle e manipular conforme desejarem. Além disso, a turbina é responsável por a cada 5 segundos, enviar uma nova mensagem ao monitoramento com os resultados do ultimo 'loop' de geração. Assim, nossa simulação nada mais é, do que uma troca de mensagens entre turbinas e monitoramento a cada 5 segundos, repassando seus detalhes de geração e tomando diferentes ações que ja forma pré-definidas.

Um importante ponto a salientar é que existem 2 tipos possíveis de comunicação entre turbinas e monitoramento, depende da arquitetura dos desenvolvedores da turbina. Este pode ser feito em (1) batches ou em (2) real-time. Assim, primeiramente em real-time, teriamos um socket aberto que a todo instante mandaria dados sobre todos os sensores da turbina. Como deve-se imaginar, apesar de ser o melhor tipo de comunicação para coleta de dados e decisões, acaba trazendo muitos problemas ao dia-a-dia do monitoramento, devido a inconsistencia na rede entre turbina e monitoramento e as falhas que afetam os controladores. Além disso, é muito comum encontrar projetos mal desenhados que acabam conflitando pacotes e informações. Por tentarmos simular o mais proximo projetos de comunicação entre turbinas que se assemalham da realidade, obtamos então pela opção que consiste em abrir um socket de comunicação com o monitoramento toda vez que um batch de dados estiver pronto para ser enviado. Assim, apesar de abrirmos e fecharmos um socket a cada troca de mensagem (de 5 em 5 segundos), estamos simulando o que acontece em turbinas a cada minuto aproximadamente.

Nesse caso, em nosso projeto, as turbinas estão a cada 5 segundos coletando os dados de vento e geração de energia - podendo nesse meio tempo gerar falhas e até mesmo "tripar". Com os 5 segundos finalizados, as turbinas então enviam uma grande mensagem para o monitoramento monstrando a média das informações geradas no ultimo 'loop'. No caso da nossa simulação, geramos os dados somente 1 vez e esperamos 5 segundos para gera-los novamente, aproximando nossa simulação do modelo real mais confiante.

É também importante salientar que foi adicionado uma pequena camada de segurança opcional no projeto. Visto que não teriamos tempo para uma versão complexa, ao iniciar uma turbina, é possível utilizar um serial padrão pré-definido para garantir que a comunicação entre turbina e monitoramento seja 'verdadeira' - evitando com que

terceiros iniciem a comunicação com o monitoramento através de IDs de turbinas falsas. A implementação está 100% completa porém bem simples e opcional. Durante a sessão de instalação e utilização é detalhado a forma com que deve ser utilizada.

Protocolo de comunicação

TCP e Cliente-Servidor

Durante o design do projeto, escolhemos por utilizar uma comunicação TCP e arquitetura cliente-servidor. Primeiramente TCP pois toda a troca de mensagem entre a turbina e o monitoramento deveria ter garantia de chegada. No mundo real, na qual essa simulação foi baseada, o modelo é hibrido entre TCP e UDP, porém para facilitarmos as coisas, utilizamos somente TCP, assim, o monitoramento sempre tem a garantia de chegada da mensagem, mesmo com um pequeno delay.

Sobre a arquitetura de cliente-servidor, a escolha foi mais simples/facil. Todas as turbinas possuem o mesmo nível hierarquico e precisam responder ao monitoramento sobre o seu status, afim de serem otimizadas para geração do vento. Ao utilizarmos a arquitetura escolhida, fica mais fácil essa comunicação 'top-down' entre todos os agentes, visto que arquiteturas distribuidas não fazem sentido para o que queremos alcançar.

Detalhes do Protocolo

Existem 5 tipos de mensagens possíveis no protocolo definido entre a turbina e o monitoramento. Vale salientar que para definição do nosso protocolo, optamos por utilizar o length da mensagem como forma de categorizar a mesma. Assim, separamos todas as formas de comunicação pelo length. O tipo de mensagem também varia. Temos mensagens que são codificadas em bytes, porém quando enviamos uma mensagem turbina -> monitoramento, geralmente estamos falando de formado String contendo numeros e letras. Ja a mensagem de retorno monitoramento -> turbina são códigos decimais simples, para que a mensagem seja leve e retorne rapidamente a turbina

1. A primeira, e mais simples de todas são mensagens de length 42B, um simples request para cancelamento da comunicação. Além disso, também desabilitamos a thread que está responsável por cuidar daquele request. Liberando o espaço na memoria para o monitoramento.

A mensagem é um simples EXITWTGXX - dessa forma, a thread que recebe essa mensagem ja sabe identificar que a turbina WTGXX deve ser desligada do monitoramento e seu socket fechado de vez.

Lembrando que essa mensagem é enviada após qualquer outra mensagem abaixo também ter sido enviada. Fechando o loop de comunicação entre cliente e servidor

2. Após a mensagem de cancelamento, temos a mensagem com length 38B e 47B - as duas são mensagens de configuração de turbina. A única diferença é que enquanto a mensagem de 38 Bytes é uma simples flag que permite a turbina continuar enviando os dados (retornando 001 caso esteja tudo ok), a segunda (de 47 Bytes) verifica se a turbina requisitora enviou corretamente o numero de serial (uma forma de segurança na hora de configurar as turbinas). Caso isso tenha acontecido com sucesso, o monitoramento também retorna 001, caso negativo, 002 e a conexão é recusada por razões de segurança.

Assim, a turbina envia: WTGXX no caso de length 38 e WTGXX:4Y7B1N8K no caso do length 47. Com isso, o monitoramento simplismente responde '001' caso positivo e '002' caso negativo

3. Por ultimo temos a mensagem de length 51B, a principal da nossa aplicação. Ela é enviada a cada 5 segundos por todas as turbinas com o resumo dos dados gerados naquele 'loop'. Essa mensagem contem diversas informações divididads em ':', como: nome da turbina geradora, vento atual na turbina, energia gerada no ultimo loop e a quantidade de falhas que a turbina possui.

Com essas informações, o monitoramento escolhe entre diversas ações que a turbina deve tomar. As respostas possíveis são: '001' - para continuar normalmente com a operação; '002' - para desligar a turbina e esperar 10 segundos devido aos fortes ventos que podem prejudicar a turbina; '003' - para aumentar a potencia da turbina, fazendo com que esta gere mais energia. Isso se deve as boas condições de vento / vento fraco, na qual a turbina precisa se esforçar mais para manter a produção. '004' - uma turbina na qual a quantidade de falhas está aumentando, se reduz a potencia da maquina, afim de não prejudicar a geração de energia da mesma.

Assim, um exemplo de comunicação seria: turbina envia -> WTGXX:13.56:14.32:2 (nome | vento atual | energia gerada (em mw/h) | #alarmes) monitor responde -> 004 (reduz potencia pois ja tem 2 falhas nessa turbina)

Para mais detalhes sobre os thresholds utilizados, existe uma documentação no código explicando porque esses numeros foram escolhidos.

Como instalar e executar a simulação

Instalando o Python 3.x

Pelo site oficial do python: https://www.python.org/downloads/. Você consegue baixar o python. Só é necessário que esteja utilizando versão 3.4 acima

Faça o clone do projeto no git | Baixe o código fonte

Faça o clone do código através do
github: https://github.com/cobap/WindfarmNetwork.git. Caso preferir, você pode baixar a versão zip do código pelo próprio github

Utilização do programa

Para começar a testar a simulação é bem simples. Basta abrir 2 prompts de comando. Entrar no diretório em que se encontra o 'monitoramento.py' e 'turbina.py'. Após isso, basta executar

```
python(3) monitoramento.py
```

Repare que em alguns casos, o comando para executar o python3 pode ser tanto 'python' como 'python3'. Para checar a versão, basta digitar um dos dois na linha de comando e ver o resultado.

Com a execução do monitoramento, agora está na hora de iniciar as turbinas, para isso, na linha de comando preencha:

```
python(3) turbina.py [ARG1] (ARG2)
```

Aqui iniciamos a turbina, por padrão, toda turbina precisa ter como primeiro argumento um 'nome'. Esta configurado para ser WTGXX, onde WTG = Wind Turbine Generator, e XX é o numero da turbina, podendo ser qualquer numero de 01 até 99. Ja o ARG2 é opcional, sendo relevante somente caso queiramos iniciar uma conexão segura com o montoramento. Nesse caso, deve-se colocar um código pré-definido que é o "serial" da turbina. Com esse código, o monitoramento verifica se o código é valido e inicia a conexão

Do modo com que foi implementada, os códigos de segurança são limitados e devem ser verificados no código fonte do programa 'monitoramento.py'

Como ler o código

Por se tratar de um programa com apenas 2 arquivos, a leitura e compreensão de torna mais fácil. Recomenda-se iniciar pela leitura do código 'turbina.py' pois é nele que a maioria das variáveis são criadas e modificadas. É interessante entender como a turbina é criada, como funciona o processo de geração de dados (vento, energia, etc) e por ultimo verificar os tipos de mensagens que a turbina é capaz de enviar. Para cada tipo de mensagem, foi criada uma função diferente, buscando modularidade

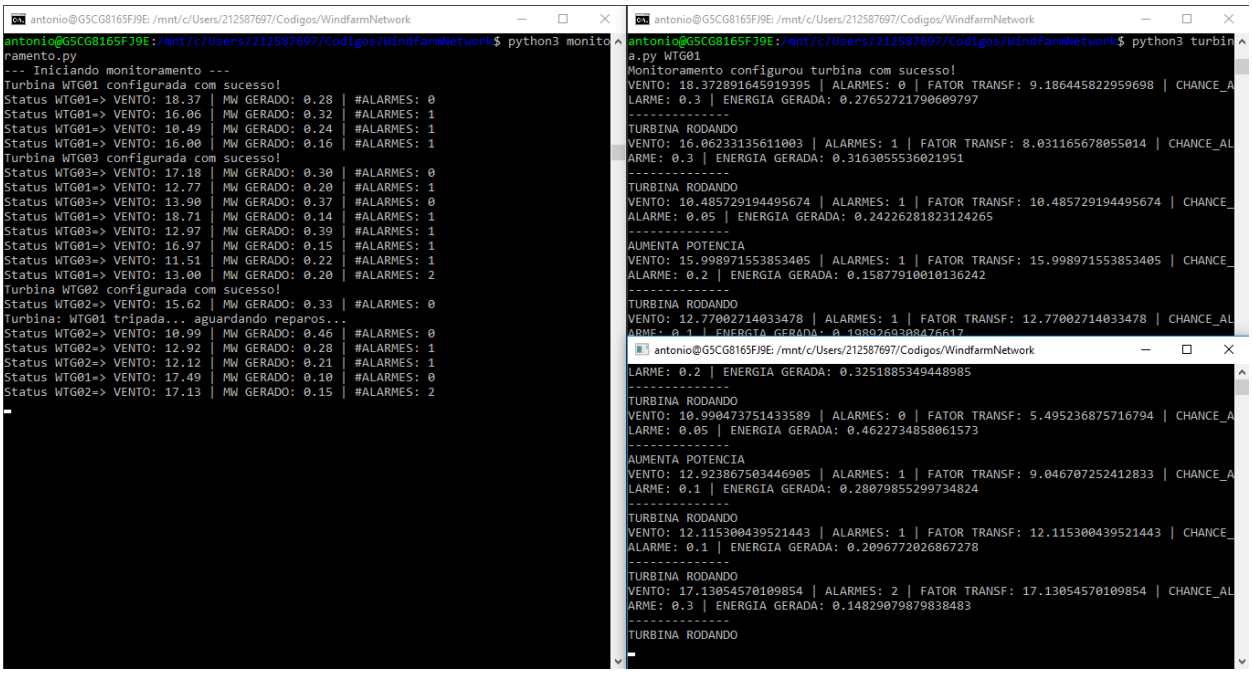
Testes e Resultados

Realizamos testes utilizando rede local (192.168.1.X) e rede aberta. Todos funcionando corretamente, apesar de um maior delay quando migravamos para rede publica.

As variáveis dos testes foram diversas: testamos utilização de parametros errados, no qual criamos multiplas causas para conter. Também lidamos com problemas da comunicação abrindo 1 socket por mensagem. Pois, devido a forma com que o programa gerava os dados e ia "dormir", o servidor ficava inumeras vezes ocioso sem necessidade. Juntando a forma com quem a implantação desses sistemas acontece na realidade, optamos então por abrir um socket por envio de mensagem.

Também testamos utilização da conexão segura entre turbina e monitor, que está funcionando corretamente apesar das limitações de códigos finitos. Também testamos com códigos "fake" e de fato a verificação é feita antes de liberar a comunicação entre cliente e servidor. Um passo que não foi implementado era gerar esses pseudo-codigos de forma (semi) automatica para que a validação pudesse ser feita em tempo real pelos dois lados

Sobre os resultados, tivemos um sucesso com a implementação da nossa simulação. Foi possível ve-la funcionando com 3 turbinas em tempo real sem nenhum problema ou lentidao. Durante alguns momentos fechavamos algumas linhas de comando e abriamos outras, e o monitoramento continuava perfeitamente, sem erros ou processamentos mal feitos.



(exemplo do programa rodando com 2 turbinas – repare que já haviamos cancelando a comunicação da WTG03)

Problemas de rede ou de implementação

Não necessariamente um problema, mas uma das dificuldades do grupo foi deixar um socket aberto por alguns segundos (15~30) enquanto os dois programas rodavam em loops gerando dados. Acabou criando uma pequena confusão de sockets sendo chamados o esquecido de ser fechados, causando grandes problemas na implantação do código. Por isso, uma de nossas soluções foi para cada troca de mensagem entre monitor e turbina, abrir um novo socket, mesmo para mensagens simples. Para um futuro projeto, seria ideal mudar a forma com que a geração e envio dos dados é feita, afim de reduzir o numero de sockets abertos durante a comunicação do programa

Outro ponto que não foi necessariamente um problema, mas poderiamos melhorar, foi a identificação do tipo de mensagem. Atualmente, só conseguimos identificar se é uma mensagem de configuração ou de envio de dados através do length (em Bytes) daquela mensagem. O programa foi desenhado de uma forma com que nenhum length se repetisse. Porém, manter esse design pode causar futuros problemas, principalmente de manutenabilidade. Para isso, o ideial seria em uma proxima versão, expandir a sintaxe das mensagens a fim de identificar previamente qual o time de mensagem que será enviada, quantos bytes devemos aguardar dela e qual o seu proposito.

Com isso, conseguimos otimizar o socket ao maximo, reduzimos o tempo gasto com transmissão de mensagens e ainda deixamos a protocolo mais robusto para eventuais mudanças e versões

Fonte

Material básico sobre redes em python: https://www.tutorialspoint.com/python/python_networking.htm

Inspiração para criação do código: https://www.techbeamers.com/python-tutorial-write-multithreaded-python-server/https://www.geeksforgeeks.org/socket-programming-multi-threading-python/

Inspiração para o código e criação do protocolo: https://realpython.com/python-sockets/