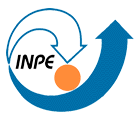
****

**INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS**

**DIVISÃO DE GEOFÍSICA ESPACIAL**

**DIVISÃO DE DESENVOLVIMENTO DE SISTEMAS DE SOLO**

**ENGENHARIA E TECNOLOGIA ESPACIAL**

**Síntese de abordagens adotadas na revisão do Projeto LEONA: Servidor, transmissão e aquisição de imagens**

2017

**Sumário**

[1 - INTRODUÇÃO 3](#__RefHeading___Toc910_742393069)

[2 - Revisão e Redefinição de tecnologias aplicadas à rede leona 4](#__RefHeading___Toc912_742393069)

[3 - Servidor nas estações leona 5](#__RefHeading___Toc920_742393069)

[3.1 - FLASK 5](#__RefHeading___Toc926_742393069)

[3.1.1 - UDP (*User Datagram Protocol*) 5](#__RefHeading___Toc932_742393069)

[3.1.2 - PROTOCOLO TCP *(Transmission Control Protocol )* 5](#__RefHeading___Toc940_742393069)

[3.1.3 - Combinações de Flask e UDP e TCP 6](#__RefHeading___Toc948_742393069)

[3.2 - TORNADO 6](#__RefHeading___Toc956_742393069)

[3.3 - Tornado e Flask 7](#__RefHeading___Toc962_742393069)

[4 - PLACA DE AQUISIÇÃO DE VÍDEO 7](#__RefHeading___Toc968_742393069)

[5 - SUBPROCESSO E MULTIPROCESSAMENTO EM PYTHON 7](#__RefHeading___Toc500_370054860)

[5.1 - Subprocessos e Multiprocessamento 8](#__RefHeading___Toc980_7423930691)

[5.2 - Sincronismo 8](#__RefHeading___Toc990_7423930691)

# INTRODUÇÃO

Este documento tem por objetivo apresentar as abordagens adotadas na revisão e redefinição de tecnologias aplicadas até maio de 2017, que aqui será nomeada de ‘.v0’, no Projeto LEONA, referentes à: captura, gravação de vídeo, transmissão ao vivo das imagens adquiridas nas estações e construção da Rede LEONA.

Este documento não possui detalhamento referente à arquitetura e funcionalidades do software da Estação LEONA, entretanto, a documentação, em desenvolvimento, está presente no arquivo ‘RedeLeonaDoc.html’ junto ao código do projeto.

# Revisão e Redefinição de tecnologias aplicadas à rede leona

No intuito de iniciar uma revisão na ‘.v0’ do projeto de software da Rede LEONA, foi realizado um estudo para elencar possíveis problemas na versão de produção do software. Os seguintes problemas foram observados:

1. Criação de uma nova estação causa uma falha no sistema operacional Windows 7;
2. Uso indevido do datador de tempo em vídeo IOTA;
3. Transmissão e acesso a somente uma câmera;
4. Requisito de captura de trinta imagens por segundo não é atendido.

Após estudo na literatura e revisão junto ao fornecedor da placa de aquisição, foram constatados que os problemas 1 e 2 eram respectivamente provenientes do uso de uma versão incompatível de software para a placa de aquisição de vídeo e não cumprimento do manual do fabricante/fornecedor. Com a instalação da versão indicada pelo fornecedor, versão 1.2.5 de 64 bits, o problema 1 foi solucionado e o problema 2 é automaticamente corrigido quando o IOTA sincroniza com mais de 3 satélites, assim como descrito no manual.

Para os problemas 3 e 4, um plano de refatoração de código foi inicializado, porém a linguagem JAVA, utilizada na ‘.v0’, mostrou-se ineficiente para a aplicação LEONA, uma vez que mesmo com o uso de artifícios da linguagem, como *threads* (programa em execução com múltiplos fluxos de execução), não foi possível atingir a taxa de gravação esperada, trinta imagens por segundo. Assim, após breve estudo na literatura a linguagem Python foi escolhida para compor a nova bateria de testes no software LEONA.

Foram realizados dois testes com a linguagem: análise da taxa de leitura da câmera pelo software; e taxa de gravação em disco das imagens adquiridas. Ambos deveriam atingir uma taxa de aquisição de trinta imagens por segundo. Para isso, o OPENcv (*Open Source Computer Vision Library*), uma biblioteca para manipulação de imagens, foi utilizada junto ao Python para tratar da aquisição das imagens e um osciloscópio realizou a medição da taxa de aquisição e gravação destas imagens. Os resultados comprovaram que a linguagem Python atende os requisitos estipulados nos testes, soluciona o problema número 4 e que a linguagem pode, então, ser utilizada para o novo software da Rede LEONA que substituirá o que atualmente se encontra nas estações. O Problema número 3 será abordado no capítulo 5 deste documento.

# Servidor nas estações leona

Uma das principais funcionalidades da Rede LEONA é fornecer um serviço de transmissão de imagens ao vivo, ao qual, se dará por meio de um portal em que o observador/pesquisador poderá ter acesso online. Neste intuito, foi inicializado a construção do servidor LEONA com a linguagem Python.

## FLASK

Flask é um *microframework web* que permite criar um servidor e é facilmente utilizado como uma biblioteca do Python. Flask foi utilizado para renderizar as imagens capturadas pela câmera, permitindo sua transmissão pela web.

Neste título entendesse o termo “transmissão das câmeras”, ou similar, até o subtitulo 3.1.3 como: transmissão da imagem capturada de somente uma câmera que é duplicada.

### **UDP (***User Datagram Protocol***)**

O UDP (em português: Protocolo de Datagrama de Usuário) é um protocolo de comunicação que se baseia no envio de pacotes de informações, é muito conhecido na literatura para realizar transmissão ao vivo e *broadcast (*transmissão para múltiplos clientes*)* de dados. O protocolo não possui garantia de verificação erros ou ordem de envio dos dados.

No protoclo UDP o tamanho máximo de transmissão de dados é de 65507 bytes, ou o equivalente a uma imagem de tamanho 320x240, muito inferior ao padrão das câmeras utilizadas na Rede LEONA, 720x480. Para garantir a transmissão ao vivo das câmeras, as imagens adquirias foram redimensionadas em um pacote menor, metade de seu tamanho, enviadas, e redimensionadas.

Foram realizados testes para verificar a experiência dos usuários durante a transmissão ao vivo. No ambiente de teste foi possível realizar a transmissão das câmeras com acesso simultâneo de setenta usuários no servidor e realizar a gravação de uma câmera com uso de *threads*, porém, devido ao redimensionamento das imagens a qualidade da transmissão ao vivo recebida pelo observador seria prejudicada, o que inviabilizou o uso do protocolo UDP.

### **PROTOCOLO TCP *(Transmission Control Protocol )***

Assim como o UDP, o protocolo TCP (em português: Protocolo de Controle de Transmissão) também é um protocolo de comunicação, o que o diferencia é a capacidade de controlar o tamanho dos pacotes enviados e verificar se o que foi recebido é igual ao que foi enviado. Por causa destas características o protocolo foi escolhido para substituir o UDP.

No protocolo TCP, cada nova conexão representa um novo caminho de conexão entre cliente, que recebe o dado, e servidor, que envia o dado, aberto (*socket*), o que resulta em novo processo, em nossa aplicação uma *thread,* gerando consumo gradativo de CPU.

Partindo do requisito de um número máximo de quinze observadores, e priorizando a taxa de gravação de trinta imagens por segundo, testes foram realizados para analisar o consumo de CPU durante o cumprimento destes dois requisitos. Mesmo com a otimização do código, as máquinas atuais das Estações LEONA, as mesmas utilizadas durante os testes, não suportariam mais do que três conexões simultâneas sem perder imagens durante a gravação, requisito de maior prioridade. Não atendendo os requisitos dos testes, o protocolo foi inviabilizado para esta aplicação.

### Combinações de Flask e UDP e TCP

Vale ressaltar, a fim de disseminar conhecimento, testes de resultado positivo e negativo durante este desenvolvimento e algumas combinações e tentativas feitas com os protocolos e soluções apresentados neste capítulo.

* Uso de dois Flasks, um como cliente e um como servidor, em máquinas diferentes para distribuir o consumo de CPU: aumento de três para seis usuários com TCP; não resolve, em UDP ou TCP, perda de imagens na gravação;
* Cliente e servidor rodando em máquinas diferentes, não ajudou de forma significativa na redução do consumo de CPU.

## TORNADO

Tornado é um *web framework* assíncrono que, de forma otimizada, trata as conexões recebidas usando protocolos de comunicação como TCP, Websocket.

Neste capítulo entendesse o termo “transmissão das câmeras”, ou similar, até o subtitulo 3.2.1, como: transmissão da imagem de duas câmeras, cada uma conectada a uma placa de vídeo. Abordagem adotada para simular o envio de imagens de duas câmeras, uma com IOTA e outra sem, conforme arquitetura proposta por *stakeholder.*

## Tornado e Flask

Nesta aplicação o Tornado foi utilizado para disponibilizar em uma porta do endereço IP do servidor LEONA a transmissão ao vivo das câmeras. Este procedimento foi utilizado para aproveitar a capacidade da solução Tornado em otimizar a transmissão de dados com métodos assíncronos e permitir que o Flask ficasse responsável somente por receber os comandos de movimento das câmeras, gravação das imagens e acesso de observadores a transmissão.

Com esta arquitetura de software foi possível atender os requisitos de gravação, trinta imagens por segundo, e o acesso de quinze observadores simultâneos. Esta versão serviu de base para a arquitetura de software adotada nesta nova versão da Rede LEONA (.v1), faltando a resolução do problema 3

# PLACA DE AQUISIÇÃO DE VÍDEO

No intuito de resolver o problema 3 descrito neste documento, Iniciou-se um estudo da *Application Programming Interface (*API), em tradução para o português: interface de programação de aplicativos, disponibilizada pelo fabricante da placa de aquisição de imagens. O requisito era obter imagens de duas câmeras utilizando a mesma placa.

A API é escrita na linguagem C/C++ e necessita de uma *Dynamic-link library* (biblioteca de vínculo dinâmico) ou mais conhecida por DLL, que possui funções como a escolha de câmeras para aquisição de imagens.

Em Python é possível fazer a leitura de funções escritas na linguagem C/C++, mas para isso o código ou a DLL deve ter sido escrita, por exemplo, em Cython, que é uma linguagem de programação para simplificar a escrita de módulos de extensão para Python, ou soluções similares, o que permite que a linguagem Python possa interpretar estas funções/métodos ou classes importadas, assim, uma vez que a DLL disponibilizada junto a placa de aquisição de imagens é um código fechado e criptografado, não foi possível modificar o código interno para o padrão necessário, sendo necessário buscar soluções em Python para a leitura de duas câmeras usando a placa disponível no projeto.

# SUBPROCESSO E MULTIPROCESSAMENTO EM PYTHON

Após uma sequência de testes, foi identificado que para a placa de aquisição de imagens fazer a leitura de mais de uma câmera é preciso iniciar um processo separado da sequência principal do código de leitura das câmeras, o que resolve o problema 3 descrito neste documento. Em Python pode-se iniciar um processo de diferentes maneiras, como, por exemplo: *threads*, subprocesso e multiprocessamento.

## Subprocessos e Multiprocessamento

Subprocessos em Python permitem que um *script* seja iniciado sem pertencer ao mesmo espaço de memória do *script* que o inicializou. Já o multiprocessamento permite que de modo análogo as threads os pacotes ou bibliotecas inclusas no script Python possam ocorrer localmente ou de modo remoto, ambos artifícios pertencem a biblioteca padrão do Python.

O subprocesso foi utilizado junto ao servidor Tornado e Flask para inicializar a leitura das câmeras, apesar de realizar a leitura das duas câmeras, atendendo a arquitetura proposta da Rede LEONA, ao iniciar um subprocesso não é possível enviar ou receber dados entre os *scripts*, pois cada um é iniciado como um programa diferente.

No multiprocessamento é possível estabelecer uma via de comunicação entre *scripts* diferentes por meio do uso de *Queues,* modulo que pertencente a biblioteca padrão do Python que permite a troca de dados de forma segura e sem perdas, entre diferentes processos. Com o use deste artifício, a ‘.v2’ da Rede LEONA foi escrita, agora atendendo a dois requisitos fundamentais: Transmissão de duas câmeras e acesso de quinze observadores na estação.

Testes mais detalhados, para garantir a taxa de aquisição de trinta imagens por segundo, demostraram que esta combinação de artifícios causa uma grande variação no consumo de CPU e memória RAM, que as atuais máquinas disponíveis nas Estações LEONA, não são capazes de estabilizar, o que faz com que esta solução não atenda o requisito na gravação das imagens em disco, pois perde quadros.

## Sincronismo

Para resolver o problema de perda de imagens durante a gravação é necessário permitir que a leitura e escrita de dados trabalhem de forma síncrona, criando um meio de comunicação entre os scripts Python de gravação e transmissão de imagens. A solução adotada, que descreve a ‘.v3’ da Rede LEONA, foi tratar e controlar ambos pelo script Python responsável pela leitura das câmeras e que fornece as imagens ao servidor Tornado. Esta troca de dados é feita com uso de uma *queue* que permite o controle do tamanho máximo de dados que é disponibilizado ao servidor Tornado por segundos, que por sua vez também acessa esse dado em uma determinada frequência, trinta vezes por segundo, seguindo o padrão de gravação descrito no requisito da Estação LEONA.

Para otimizar o consumo de CPU e memória RAM, as funções que antes eram controladas pelo servidor Flask, na ‘.v2’ da Rede LEONA foram reescritas no servidor Tornado, permitindo que qualquer outra solução possa ser adotada no Servidor Central da Rede LEONA.