Experimento 09:HTTP

*RIP

1.º Fabrício de Oliveira Barcelos
Dept. de Engenharia Elétrica (FT-ENE)
Universidade de Brasília (UnB)
Brasília, Brasil
fabriciobarcellos01@gmail.com

2.º Pedro Henrique Dornelas Almeida
Dept. de Engenharia Elétrica (FT-ENE)
Universidade de Brasília (UnB)
Brasília, Brasil
phdornelas.almeida@gmail.com

***Resumo*—Neste experimento serão estudados os conceitos do mecanismo de funcionamento da camada de aplicação HTTP e os seus mecanismos de funcionamento.**

***Index Terms*—Camada de aplicação, HTTP, Wireshark.**

I. Objetivos

Este documento tem como objetivo principal estudar os conceitos envolvidos para o funcionamento da camada de aplicação HTTP, a partir da utilização do software *WireShark*, ao abrir um endereço HTML no navegador.

II. Introdução Teórica

Neste momento serão apresentados os conceitos teóricos necessários para o completo entendimento do experimento e ser possível analisar os resultados com clareza.

Para o experimento será necessário entender sobre a camada de aplicação, esta que é a mais próxima ao usuário e é responsável pela comunicação fim-a-fim. Depois, serão estudados dois protocolos da camada de transporte que atuam juntos a camada de aplicação, do protocolo HTTP.

A. Camada de Aplicação.

Camada de aplicativo é um termo usado em redes de computadores para designar uma camada de abstração que contém protocolos que realizam a comunicação ponta a ponta entre os aplicativos. No modelo OSI, é a sétima camada. É responsável por fornecer serviços para aplicativos para distinguir a existência de comunicação em rede entre processos de diferentes computadores. No modelo TCP / IP, é a quinta camada (segundo o autor, pode ser a quarta camada), e também inclui a camada de apresentação e a camada de sessão no modelo OSI.

É a camada mais próxima do usuário e é responsável por quando os clientes acessam *e-mails*, páginas da web, mensagens instantâneas, logins remotos, videoclipes, videoconferências, etc. A arquitetura do aplicativo permite que os usuários acessem essas funções. Portanto, existem três tipos de arquiteturas:

- Arquitetura cliente-servidor;
- Arquitetura P2P;
- Arquitetura híbrida cliente-servidor e P2P;

O protocolo da camada de aplicação funciona com o protocolo da camada de transporte (TCP / IP e UDP). Portanto, os principais protocolos de aplicação são: TELNET, FTP, TFTP, SMTP, POP, IMAP, DNS, HTTP, HTTPS, RTP, MIME e TLS. Como outras camadas do modelo, o protocolo da camada de aplicação também depende do protocolo da camada inferior para transmissão de dados pela rede - os dados do protocolo de aplicação são encapsulados no protocolo da camada inferior.

B. Mecanismo de funcionamento do protocolo HTTP.

HTTP é um protocolo que permite obter recursos (como documentos HTML). É a base de todos os protocolos de troca de dados e cliente-servidor na Web, o que significa que a solicitação é iniciada pelo destinatário (geralmente um navegador da Web) e um documento completo é reconstruído a partir dos diferentes subdocumentos obtidos. , instruções de layout, imagens, vídeos, *scripts*, etc.

O cliente e o servidor se comunicam trocando uma única mensagem em vez de um fluxo de dados. A mensagem enviada pelo cliente (geralmente um navegador da web) é chamada de solicitação, também chamada de solicitação, e a mensagem enviada pelo servidor como resposta é chamada de resposta. O projeto do HTTP foi iniciado em 1990 é extensível e evoluiu ao longo do tempo. Atuando na camada de aplicação e é enviado sobre o protocolo TCP, ou em uma conexão TCP criptografada com TLS, embora qualquer protocolo de transporte confiável possa ser usado. Como este é um protocolo extensível, ele é usado não só para buscar documentos de hipertexto, mas também imagens e vídeos ou publicar conteúdo em servidores, como nos resultados de formulário HTML. O HTTP também pode ser usado para buscar partes de documentos para atualizar páginas da Web sob demanda.

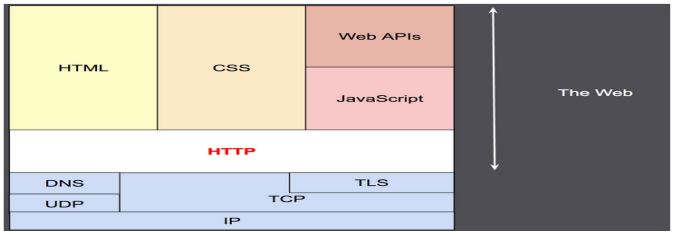

Figura 1. Cabeçalho HTTP

C. Componentes de sistemas baseados em HTTP

HTTP é um protocolo cliente-servidor: a solicitação é enviada pela entidade, o agente do usuário (ou um agente em seu nome). Na maioria das vezes, o agente do usuário é um navegador da web, mas também pode ser qualquer coisa, como um robô que varre a web para preencher e manter um índice de mecanismo de pesquisa e coletar informações.

Cada solicitação individual é enviada ao servidor, e o servidor processa a solicitação e fornece o resultado, que é chamado de resposta. Entre a solicitação e a resposta, existem várias entidades (chamadas coletivamente de *proxies*) que realizam diferentes operações e agem como *gateways* (intermediários) ou caches.

Na verdade, existem muitos outros computadores processando solicitações entre o navegador e o servidor: roteadores, modems, etc. Devido ao uso do modelo da camada da Web, essas funções estão ocultas na camada de rede e na camada de transporte. O HTTP está no topo da camada de aplicação. Apesar de ser importante diagnosticar problemas de conectividade, as camadas subjacentes são irrelevantes para a descrição do HTTP.

D. HTTP e conexões.

A conexão é controlada na camada de transporte, portanto, está basicamente fora do escopo do controle HTTP. No entanto, o protocolo de transporte que o HTTP não precisa usar é baseado em conexão, apenas requer que seja confiável ou não perca mensagens (pelo menos não haverá erros). Entre os dois protocolos de transmissão mais comuns na Internet, o TCP é confiável, mas o UDP não. Portanto, o HTTP usa o padrão TCP baseado em conexão, embora o uso de conexões nem sempre seja obrigatório.

No protocolo HTTP / 1.0, uma conexão TCP é aberta para cada par solicitação / resposta trocada, o que traz duas desvantagens principais: a abertura da conexão requer múltiplas idas e voltas de mensagens, portanto a velocidade é muito lenta, mas no envio de mensagens maior eficiência. Mais ou mais frequentemente: ”Conexões ativas”são mais eficientes do que ”conexões frias”(envio de poucas mensagens ou com menos frequência).

Para resolver essas deficiências, o protocolo HTTP / 1.1 introduz o conceito de linha de produção (ou pipeline) (que se mostrou difícil de alcançar) e conexão persistente: você pode usar a parte do cabeçalho *HTTP Connection* para controlar a conexão TCP feita abaixo. O HTTP / 2.0 foi mais longe, multiplexando várias mensagens em uma única conexão, ajudando assim a manter a conexão mais quente e eficiente.

E. O que pode ser controlado pelo HTTP

Com o tempo, a natureza extensível do HTTP permite mais controle e funcionalidade na Internet. Desde o início da história do HTTP, o armazenamento em cache e a autenticação são recursos suportados. Em contraste, a função de relaxar as restrições da fonte foi adicionada na década de 2010.

- **Cache**: O método de armazenamento em cache de documentos pode ser controlado via HTTP. O servidor pode instruir o agente e o cliente sobre o conteúdo e o tempo a ser armazenado em cache. O cliente pode instruir o agente de armazenamento em cache intermediário a ignorar o documento armazenado.
- **Relaxamento das restrições na origem**: Evitar bisbilhoteiros e outros invasores de privacidade, o navegador impõe estritamente a separação dos sites, e apenas páginas da mesma fonte podem acessar todas as informações da página. Apesar dessa restrição ser um fardo grande aos servidores, os cabeçalhos HTTP podem relaxar essa separação estrita no lado dos servidores, permitindo que um documento seja composto por várias fontes de informação em outros domínios (e pode até ter razões específicas de segurança para se fazer isso), como um tecido de retalhos.
- **Autenticação**:Certas páginas podem ser protegidas para que apenas alguns usuários possam acessá-las. Você pode usar o cabeçalho *WWW-Authenticate*etc. para fornecer autenticação básica via HTTP ou usar *cookies* HTTP para configurar uma sessão específica.
- **Proxy e tunelamento** Os servidores e / ou clientes geralmente estão localizados na intranet, ocultando seus endereços IP reais de terceiros. As solicitações HTTP usam um *proxy* para contornar essa barreira na rede. No entanto, nem todos os proxies são *proxies HTTP*. Por exemplo, o protocolo SOCKS é executado em um nível inferior. Esses agentes podem lidar com outros protocolos, como ftp.
- **Sessões** Usando cookies HTTP, ele permite que você vincule solicitações ao status do servidor. Embora o protocolo HTTP básico não mantenha o estado, ele cria uma sessão. Isso não é útil apenas para carrinhos de compras de *e-commerce*, mas também para qualquer site que permita respostas personalizadas no nível do usuário.

F. Fluxo HTTP

Quando o cliente deseja se comunicar com o servidor, seja um servidor de terminal ou um agente, ele executará as seguintes etapas:

1) Abrir uma conexão TCP: a conexão TCP será usada para enviar uma ou mais solicitações e receber respostas. Os clientes podem abrir novas conexões, reutilizar as conexões existentes ou abrir várias conexões com o servidor.
2) Enviar ma mensagem HTTP: as pessoas podem ler mensagens HTTP (antes de HTTP / 2.0). Com HTTP / 2.0, essas mensagens simples são encapsuladas na estrutura, tornando-as ilegíveis diretamente, mas o princípio permanece o mesmo.

```
GET / HTTP/1.1
Host: developer.mozilla.org
Accept-Language: fr
```

Figura 2. Solicitação HTTP

3) Ler a resposta do servidor:

```
HTTP/1.1 200 OK
Date: Sat, 09 Oct 2010 14:28:02 GMT
Server: Apache
Last-Modified: Tue, 01 Dec 2009 20:18:22 GMT
ETag: "51142bc1-7449-479b075b2891b"
Accept-Ranges: bytes
Content-Length: 29769
Content-Type: text/html

<!DOCTYPE html... (here comes the 29769 bytes of the requested web page)
```

Figura 3. Resposta do servidor HTTP

4) Fechar a conexão ou reutilize-a para solicitações futuras.

Se o pipeline estiver ativado, várias solicitações podem ser enviadas sem receber a primeira resposta completamente. Os fatos comprovam que a linha de montagem HTTP é difícil de implementar na rede existente, pois na rede existente coexistem o software antigo e a versão moderna. A linha de montagem HTTP tem sido substituída no HTTP/2.0 com multiplexação mais robusta de requisições dentro de um quadro (*frame*).

G. Requisição HTTP

Para entender melhor uma requisição HTTP, foi utilizado o seguinte exemplo:

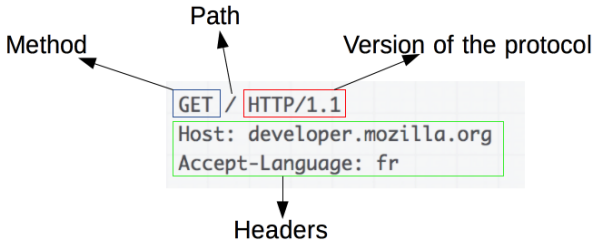

Figura 4. Requisição HTTP

- Os métodos HTTP geralmente são verbos (como GET, POST, DELETE, PUT, etc.) ou substantivos (como OPTIONS ou HEAD) para definir a operação a ser realizada pelo cliente. Normalmente, os clientes consomem recursos (usando GET) ou postam dados de formulários HTML (usando POST), embora em outros casos mais operações possam ser necessárias.
- A maneira de encontrar o recurso; a URL do recurso, que não tem elemento de contexto, por exemplo, nenhum protocolo de protocolo (http: //), domínio de domínio (aqui developer.mozilla.org) ou porta porta TCP (aqui indicada por 80) está Oculto porque é o número da porta padrão)
- A versão do protocolo HTTP.
- Um cabeçalho opcional que contém informações adicionais sobre o servidor.
- Para alguns métodos, como POST, é semelhante ao corpo da resposta ou ao corpo dos dados que contém o recurso solicitado.

E a resposta da requisição:

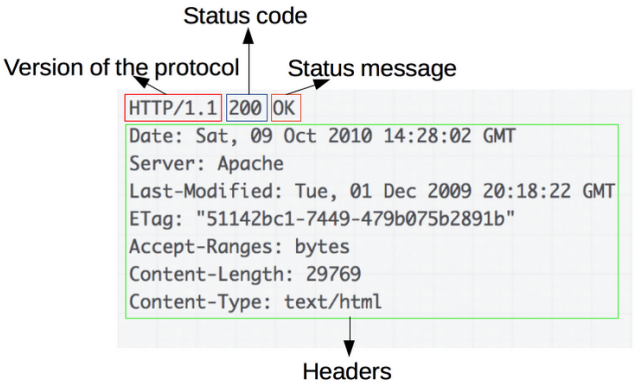

Figura 5. Resposta HTTP

- Eles seguem a versão do protocolo HTTP.
- Código de status, indicando se a solicitação foi bem-sucedida e por quê.
- Mensagem de status, uma breve descrição informal do código de status.
- Cabeçalhos HTTP, como os da solicitação. (Opcional)
- O assunto com os dados de recursos solicitados.

III. Descrição das Atividades e Análises

Após iniciar o *Wireshark*, deve-se digitar o seguinte endereço no navegador http://gaia.cs.umass.edu/wiresharklabs/HTTP-wireshark-file1.html , e então pode-se analisar o que aconteceu.

A. Parte 1

Filtrando por pacotes http, e após selecionar Analyze → Follow TCP Stream é possível responder as seguintes perguntas abaixo. Os pacotes analisados podem ser vistos como a figura a seguir:

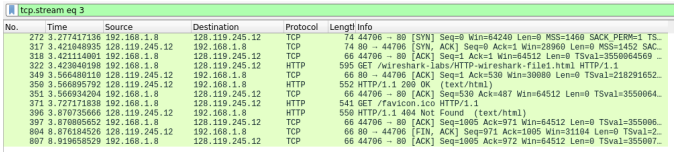

Figura 6. Pacotes Wireshark

Após selecionar Analyze → Follow TCP Stream é possível ver em vermelho os pacotes enviados pelo *browser*, e as informações em azul os pacotes enviados pelo servidor:

Figura 7. TCP Stream

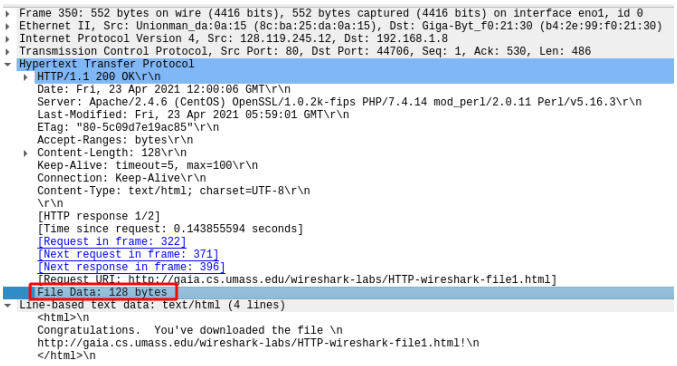

Figura 8. Bytes

1) Qual a versão seu *browser* está rodando do HTTP, 1.0 ou 1.1? E qual versão roda o servidor?
   O *browser* está rodando na versão HTTP 1.1 e o servidor também, é possível ver essas informações nas figuras 6 e 7.
2) Que línguas (se houver alguma), seu *browser* indica ao servidor, que pode aceitar?
   Na figura 7 é possível ver nas informações do primeiro pacote, no campo ”Accept-Language: en-US, en; q=0.9, pt-BR; q=0.8, pt; q=0.7”, logo, o *browser* aceita Inglês e Português.
3) Qual o endereço IP de seu PC? E do servidor gaia.cs.umass.edu?
   É possível ver da figura 6 que o IP do computador é 192.168.1.8 e o IP do servidor é 128.119.245.12.
4) Ao fazer a requisição, qual é o *status code* retornado pelo servidor?
   O *status code* é 200 OK, pode ser encontrado na figura 6.
5) Quando foi a última vez que o arquivo HTML requisitado foi modificado no servidor?
   A última modificação pode ser encontrada na figura 7 pela primeira informação em azul, no campo ”Last-Modified: Fri, 23 Apr 2021 05:59:01 GMT”.
6) Quantos bytes de conteúdo foram retornados pelo servidor, ao fazer-se a requisição?
   Para ver com clareza, foi inspecionado o pacote de resposta:

E é possível ver que a resposta contém 128 bytes.

B. Parte 2

Após limpar o cache do navegador é possível visitar a página http://gaia.cs.umass.edu/wiresharklabs/HTTP-wireshark-file1.html novamente, e após ela carregar, deve-se requisitar a página novamente, e então é possível ver que os pacotes enviados e recebidos ficaram da seguinte maneira:

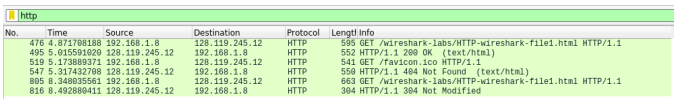

Figura 9. Requisições e Respostas - Parte 2

7) Inspecione o conteúdo da primeira resposta do servidor. Qual o *status code* retornado pelo servidor em resposta a requisição? O servidor retorna explicitamente o conteúdo do arquivo HTML? Como é possível verificar?
   Após inspecionar o conteúdo da primeira resposta é possível ver:

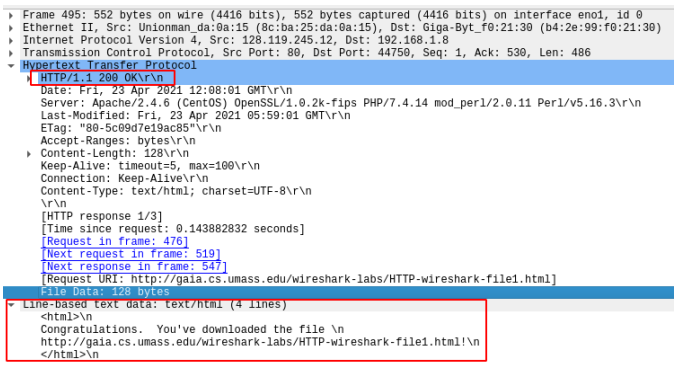

Figura 10. Conteúdo primeira resposta

Então é possível dizer que o *status code* retornado pelo servidor é ”200 OK”, e o servidor retorna o conteúdo explicitamente. É possível verificar no campo marcado pela imagem acima.

8) Qual o *status code* retornado pelo servidor em resposta à segunda requisição? O servidor explicitamente retorna o conteúdo do arquivo pela segunda vez? Explique.
Após inspecionar o conteúdo da segunda requisição foi possível ver:

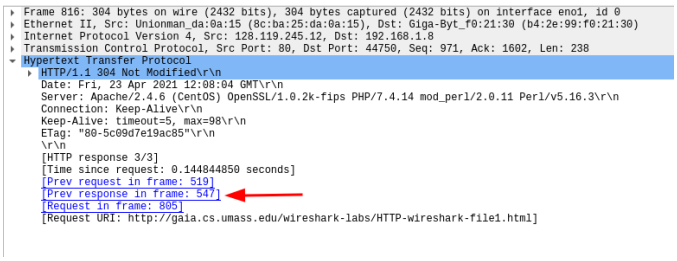

Figura 11. Conteúdo segunda resposta

Então é possível ver que o *status code* retornado é ”304 Not Modified”, e o servidor não retorna explicitamente o conteúdo do arquivo. Isto ocorre porque como o arquivo não foi modificado e houve uma requisição a pouco tempo, o arquivo está em cache no navegador, logo, ele verifica se o arquivo não foi modificado, e como há essa constatação, o arquivo que o navegador tem é o mais atualizado, assim, não há a necessidade de transferir o arquivo novamente.

C. Parte 3

Aqui deve-se acessar uma página web com conteúdo extenso, é possível encontrar a página visitada no endereço: http://gaia.cs.umass.edu/wireshark-labs/HTTP-wireshark-file3.html .

Então, analisando no wireshark, foi possível ver:

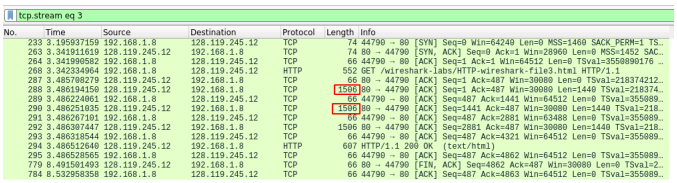

Figura 12. Parte 3

9) Quantas requisições GET foram enviadas pelo *browser*? Como é possível ver na figura 12 , foi enviado somente uma requisição do tipo GET.
10) Quantos segmentos TCP contendo dados HTTP foram enviados em resposta ao GET?
Como também é possível ver na figura 12 foram enviados 7 pacotes TCP contendo os dados pedidos.

D. Parte 4

Aqui deve-se verificar sobre a autenticação necessária a uma página com esse tipo de requisito. Para isso deve-se acessar o site http://gaia.cs.umass.edu/wireshark-labs/protected_pages/HTTP-wireshark-file5.html, e utilizando o Login: wireshark-students e senha: network , então foi possível analisar o que ocorreu.

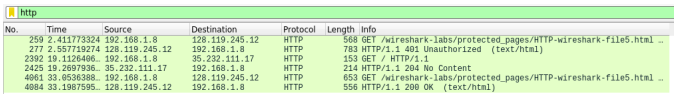

Figura 13. Parte 4 - Pacotes

11) Qual a resposta (*status code* e frase) do servidor, em resposta ao primeiro GET enviado pelo *browser*?
Inspecionando a resposta do servidor foi possível ver:

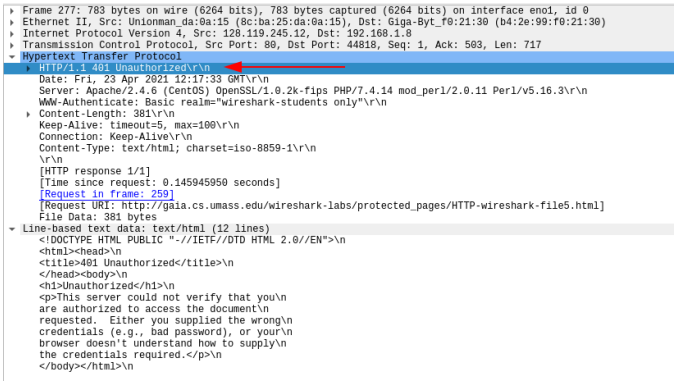

Figura 14. Primeira resposta

Então, o *status code* e frase foi ”401 Unauthorized”.

12) Quando o *browser* envia a segunda requisição GET ao servidor, que campo agora está incluído na mensagem desta requisição?
Inspecionando a segunda requisição GET enviada pelo *browser*, o campo incluindo está destacado na imagem a seguir, ele contém o usuário e senha:

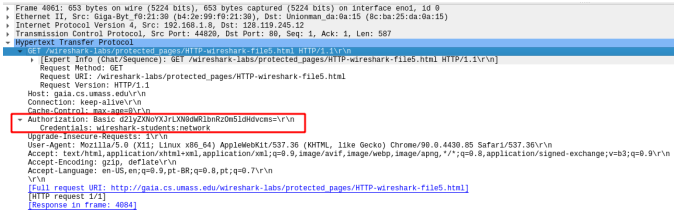

Figura 15. Segunda Requisição

IV. Conclusão

Aqui pode-se concluir que o experimento foi realizado com sucesso, podendo ver na prática tudo aquilo que foi descrito anteriormente sobre como o protocolo HTTP funcionam servindo a uma aplicação, bem como os detalhes desses protocolo. Como por exemplo, as flags de início de conexão, tamanho dos pacotes transmitidos, número de requisições, quantidade de bytes de conteúdo retornados pelo servidor, as respostas da requisições. Também foi possível concluir sobre o funcionamento do HTTP atuando na camada de aplicação.

Referências

[1] Flavio Elias de Deus. Universidade de Brasília. (Brasil) [Laboratório de arquitetura e protocolos de redes]. *Roteiro do Experimento 09:HTTP*, https://aprender3.unb.br/pluginfile.php/731309/mod_resource/content/1/

Roteiro%20Exp09%20-%20Aplica%C3%A7%C3%A3o.pdf

[2] J. F. Kurose e K. W. Ross, *Computer Networks: A Top-Down Approach. (5th ed.). Pearson Addison-Wesley, 2009.*

[3] Apresentação ”Aula 08”. Disponível em: https://aprender3.unb.br/pluginfile.php/731307/mod_resource/content/1/Aula%2009%20-%20Aplicacao.pdf