



## CAMADAS DE APLICAÇÃO E TRANSPORTE

### DEFINIÇÃO

Estudo das camadas de aplicação e transporte do modelo OSI, além da compreensão dos serviços oferecidos por cada camada. Identificação da arquitetura utilizada no desenvolvimento de aplicações, com destaque para as principais disponíveis na camada da internet. Análise dos elementos de suporte dos serviços de transporte com e sem conexão nesta camada.

### PROpósito

Compreender a influência de uma arquitetura no desenvolvimento de aplicações para redes de computadores, bem como os impactos dos diferentes serviços oferecidos pela camada de transporte no funcionamento delas.

### OBJETIVOS

Módulo 1	Módulo 2	Módulo 3	Módulo 4
Reconhecer as arquiteturas de aplicações	Identificar os principais serviços oferecidos pela camada de aplicação	Localizar os elementos da camada de transporte	Comparar os serviços oferecidos pela camada de transporte

## Módulo 1



### Reconhecer as arquiteturas de aplicações

## CAMADA DE APLICAÇÃO

Atualmente, as **redes de computadores** estão presentes no cotidiano das pessoas, permitindo a interação e a realização de diversas tarefas.



Em relação às redes de comunicação, você já ouviu falar no trabalho da **camada de aplicação**?

## NA PRÁTICA

Vamos analisar os conceitos estudados na prática? Veja o caso a seguir.

Quando realizamos uma compra com cartão de crédito ou débito em um estabelecimento comercial, é fundamental a existência de uma rede de comunicação, já que ela será o alicerce para execução da operação.

Ao inseri-lo na máquina de cartão, precisamos colocar uma senha para confirmar a operação. Tal dado é inserido no sistema por meio de um *software* executado nesta máquina.



Em qual camada este *software* é executado?

**Na camada de aplicação.**

O *software* de aplicação, também conhecido como *software* aplicativo, é nossa interface com o sistema (e, por consequência, com toda a rede de comunicação que suporta essa operação). Portanto, sempre que houver um serviço na rede, virá à mente a interface com ele.

Outros exemplos de *softwares* de aplicação:



Navegador web



Cliente de e-mail



Jogos executados em rede

Ressaltamos que a camada de aplicação é aquela de **mais alto nível** do modelo OSI\*, fazendo a interface com os usuários do sistema e realizando as tarefas que eles desejam.

**\*Modelo OSI:** O modelo OSI (*open system interconnection*) foi criado pela International Organization for Standardization (ISO) com o objetivo de ser um padrão para a construção de redes de computadores. O OSI divide a rede em sete camadas: cada uma realiza funções específicas implementadas pelo protocolo da camada. Desse modo, elas prestam serviços para a camada superior.

## ARQUITETURAS DE APLICAÇÕES

Façamos a seguinte suposição: nosso objetivo é desenvolver uma aplicação a ser executada em rede. Para criá-la, deve-se utilizar uma **linguagem de programação que possua comandos e/ou funções para a comunicação em rede**.



Na maioria das linguagens, esses comandos e/ou funções estão em bibliotecas nativas da linguagem ou criadas por terceiros.

Mas não basta conhecer uma linguagem de programação e suas bibliotecas. Antes disso, é preciso definir qual **arquitetura** terá sua aplicação. Entre as mais conhecidas, destacam-se as seguintes:

Cliente-servidor

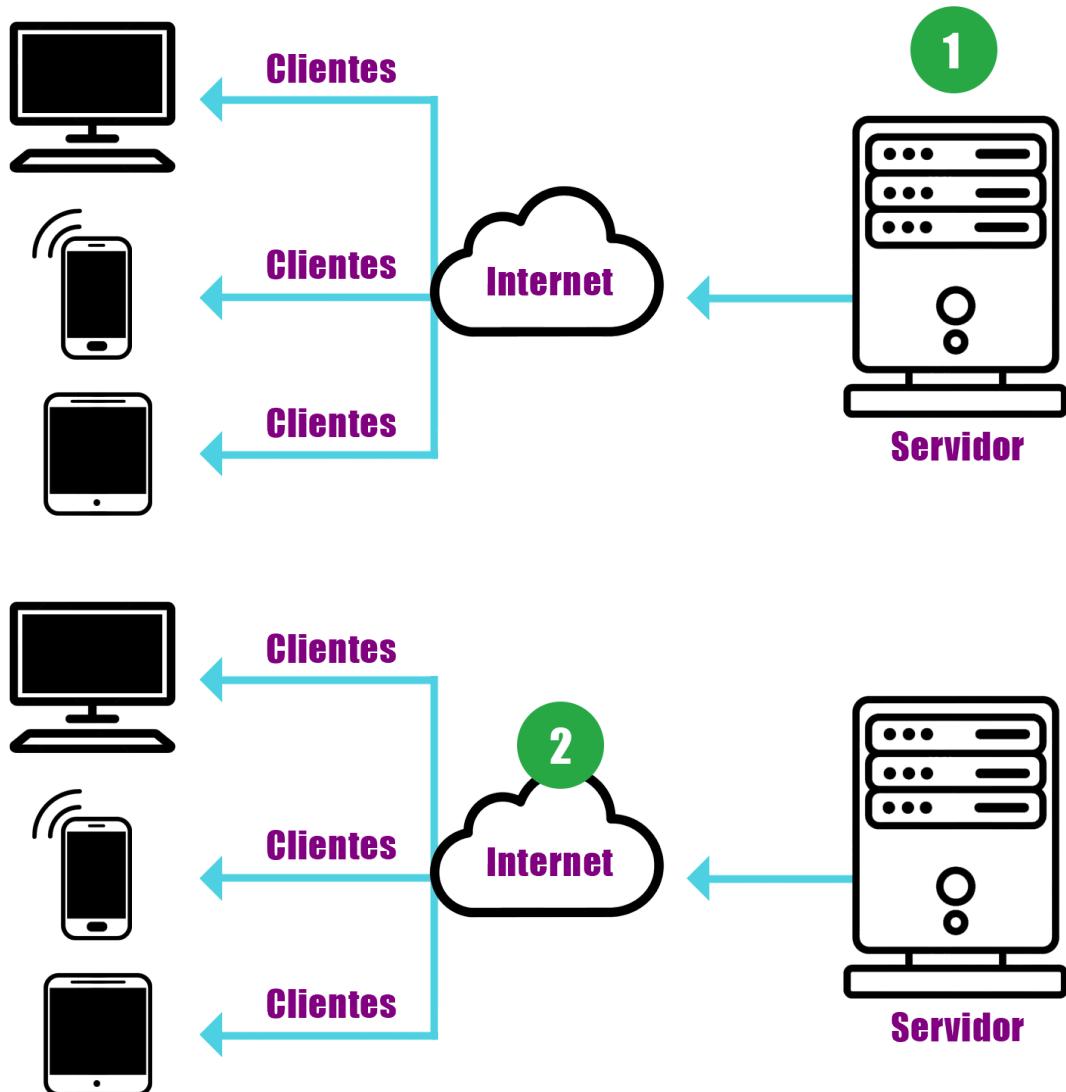
*Peer-to-peer*(ou P2P)

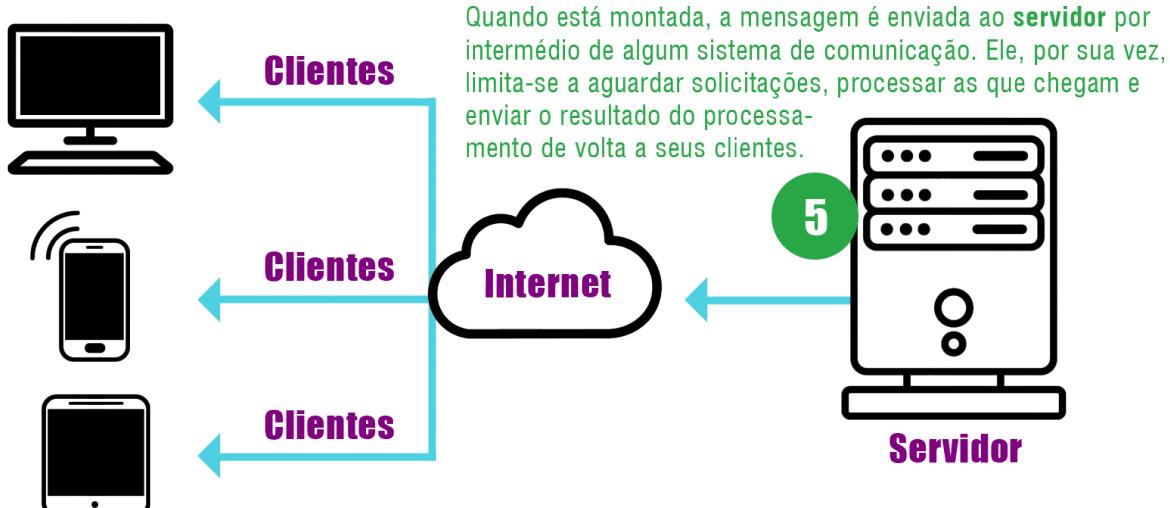
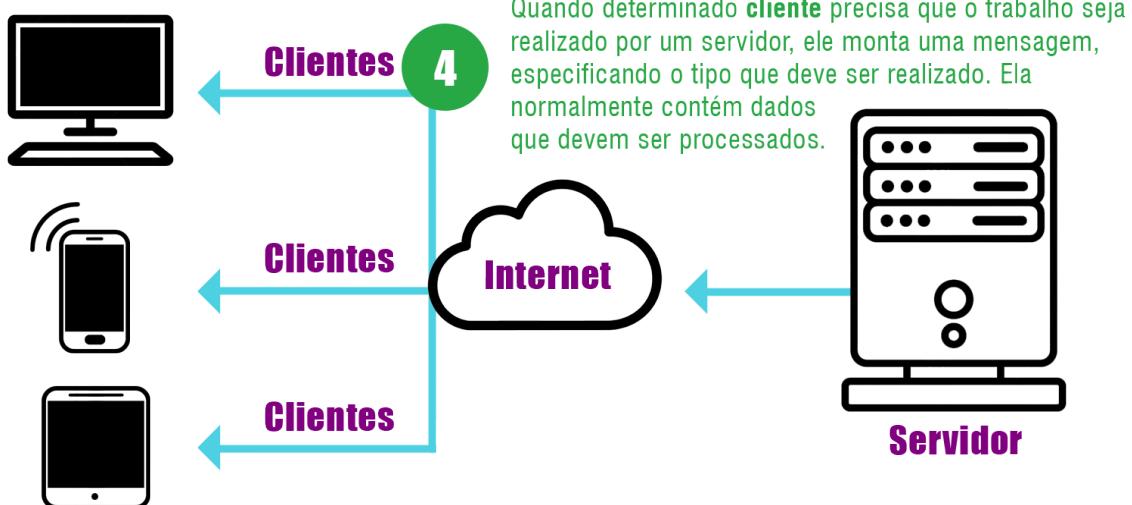
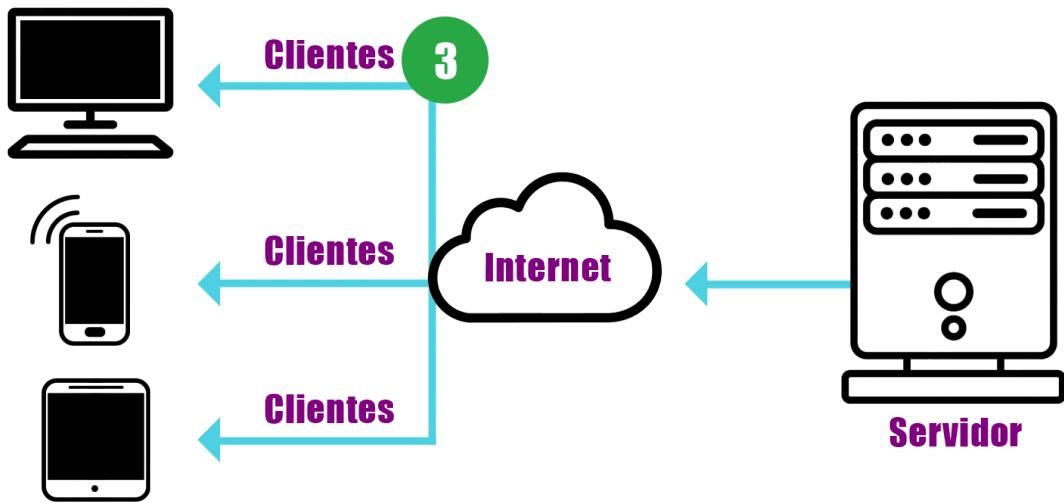


## Cliente-servidor

Nesta arquitetura, há pelo menos duas entidades: um **cliente** e um **servidor**. O servidor executa operações continuamente aguardando por requisições daquele(s).

Modelo de arquitetura cliente-servidor







Servidor

- ✓ Quando chega uma solicitação, o servidor pode:
- ✓ Atender imediatamente caso esteja ocioso;
- ✓ Enfileirar a solicitação para ser atendida mais tarde;
- ✓ Gerar um **processo-filho\*** para o atendimento da solicitação;
- ✓ Criar uma **thread\*** para esse atendimento.

**\*Processo-filho:** Um processo é um programa em execução que inclui sua região de memória, os valores das variáveis e seu contexto de *hardware*. Um processo-filho é criado quando determinado processo se duplica em memória e entrega à sua cópia (seu filho) uma tarefa a ser executada.

**\*Thread:** Linhas de execução independentes que executam concorrentemente dentro de um processo. Quando funções são executadas como *threads* dentro dele, isso é feito de forma concorrente, compartilhando objetos e variáveis.

Independentemente do momento em que uma solicitação é processada, o servidor, no final, envia ao cliente uma mensagem contendo **o resultado do processamento**.

Quem utiliza este tipo de arquitetura é a **aplicação web**.

Vamos analisar esse processo no exemplo a seguir.



Você deseja fazer uma receita especial, descobrindo, em um site, aquele prato que gostaria de preparar. Ao clicar em um *link*, ela irá aparecer. Para isso acontecer, o servidor *web* (*software* servidor do site de receitas) fica aguardando as conexões dos clientes.





Quando você clica no link da receita, seu *browser* envia uma mensagem ao servidor indicando qual delas você quer.



Ele faz então o processamento solicitado e devolve ao *browser* resultado disso (sua receita).

É muito importante compreender, de maneira prática, o funcionamento do processamento de resultados.



**Atenção!**

O que **determina** se uma entidade é **cliente** ou **servidor** é a **função desempenhada** pelo *software*, e não o tipo de equipamento.



## É fundamental saber que...

Servidores desempenham uma função muito importante; por isso, há equipamentos apropriados para eles, com **MTBF\*** alto e recursos redundantes.



O tipo de *software* instalado neste equipamento é o **responsável** por **determinar** se ele é **cliente ou servidor**.

**\*MTBF:** Do inglês *mean time between failures* (ou período médio entre falhas), esta sigla indica o tempo esperado até que ocorra uma falha no dispositivo. Quanto maior o MTBF, mais confiável um dispositivo é considerado.

Além disso, **um processo pode atuar simultaneamente como cliente e servidor.**

Voltemos ao exemplo da aplicação *web*:



Quando seu *browser* solicita a receita ao servidor *web*, aquele está atuando como cliente e este, como servidor.





Mas esse **processo nem sempre é simples**; afinal, a aplicação que executa no servidor *web* e realiza o processamento solicitado pode precisar de uma informação armazenada em um banco de dados externo.



Para obtê-la, este servidor deve enviar uma mensagem ao servidor de banco de dados solicitando aqueles de que necessita para continuar. Neste momento, ele atua como um **cliente do servidor de banco de dados**.

### *Peer-to-peer*

Enquanto existe uma distinção bem clara entre os processos que trocam informações na arquitetura cliente-servidor, na ***peer-to-peer*\*** – também conhecida como arquitetura P2P –, todos os processos envolvidos desempenham funções similares.

**\**Peer-to-peer*:** O termo *peers* surge do fato de os processos se comunicarem diretamente sem a intervenção de servidores, promovendo uma comunicação par a par (*peer-to-peer*).



Em geral, nesses sistemas, os processos não são uma propriedade de corporações. Quase todos os participantes (senão todos) são provenientes de usuários comuns executando seus programas em *desktops* e *notebooks*.

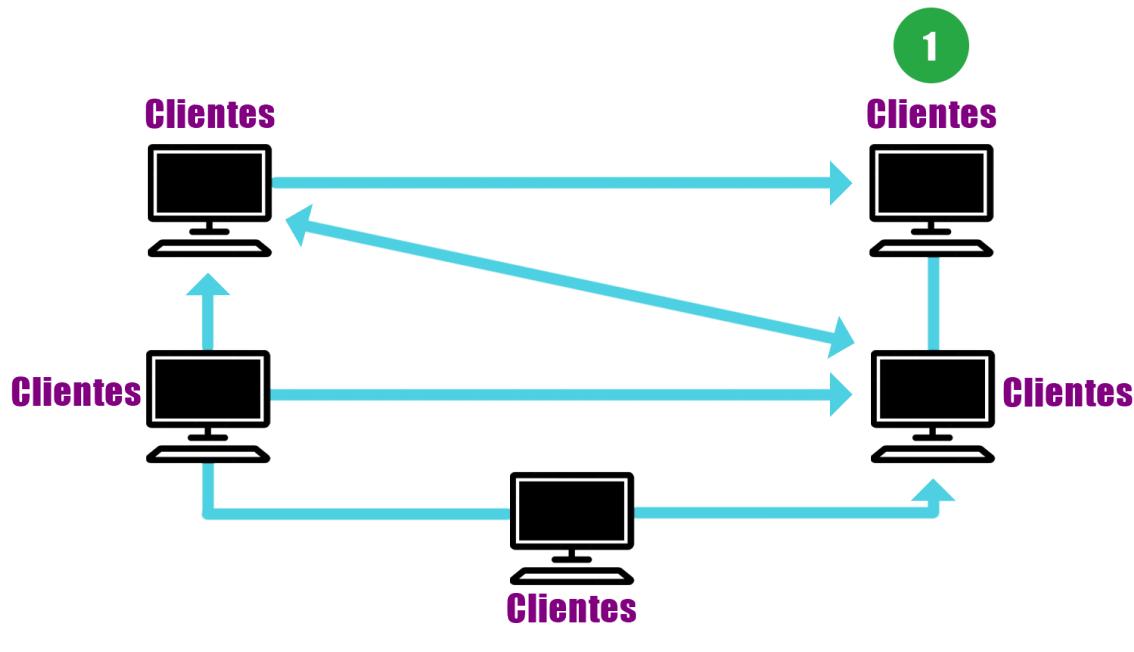


Tanto o processamento quanto o armazenamento das informações são distribuídos entre os **hospedeiros\***. Isso lhes confere maior escalabilidade em comparação à arquitetura cliente-servidor.

**\*Hospedeiro:** Também conhecido como *host*, o hospedeiro é qualquer equipamento conectado à rede capaz de trocar informações com outros equipamentos. Exemplos: computadores, roteadores, impressoras de rede, smartphones etc.

### Modelo de arquitetura *peer-to-peer*:

*Na ordem, 1, 2 e 3, o fluxo de informações e como elas se organizam.*





3

Esta arquitetura foi bastante impulsionada pelo surgimento dos computadores pessoais de alto desempenho e das redes banda larga.

## NA PRÁTICA

Vamos analisar os conceitos estudados na prática? Responda à questão a seguir.

Você conhece algum tipo de sistema de compartilhamento utilizado na internet? Em qual tipo de arquitetura ele está fundamentado?

[Ver Resposta](#)

Uma aplicação amplamente utilizada na internet é o sistema de compartilhamento de arquivos BitTorrent.

Baseado na arquitetura *peer-to-peer*, este sistema permite que seus usuários compartilhem arquivos sem haver a necessidade de eles estarem armazenados em um servidor.

## VERIFICANDO O APRENDIZADO

1. Uma arquitetura amplamente utilizada no desenvolvimento de aplicações em rede é a cliente-servidor. Sobre ela, responda:

- a) Cabe ao servidor iniciar toda negociação com seus clientes.
- b) Existe uma distinção bem clara entre clientes e servidores. Um processo não pode ser cliente e servidor simultaneamente.
- c) É a função desempenhada pelo *software* que determina se a entidade é cliente ou servidor
- d) Quando chega uma requisição de um cliente, o sistema operacional deve iniciar a execução do servidor.

### Comentário

**Parabéns!** A alternativa **C** está correta.

O que determina se a função desempenhada pela entidade é de cliente ou de servidor é seu comportamento definido pelo algoritmo executado, ou seja, pelo *software*.

2. Marque a afirmativa verdadeira no que se refere à arquitetura *peer-to-peer*.

- a) Permite uma forma de armazenamento distribuída desde as informações do sistema.
- b) Cada processo na arquitetura desempenha uma função bem definida e diferente das funções dos demais processos.
- c) Um dos maiores problemas é sua baixa escalabilidade.
- d) A comunicação entre os processos deve ser intermediada por um servidor.

### Comentário

**Parabéns!** A alternativa **A** está correta.

Na arquitetura peer-to-peer, todas as entidades desempenham a mesma função, possuindo igual capacidade de armazenamento das informações do sistema. Dessa forma, uma informação procurada pode estar com qualquer um dos participantes da rede, estando distribuída por todo o sistema.



Identificar os principais serviços oferecidos pela camada de aplicação

## PROTOCOLOS DA CAMADA DE APLICAÇÃO

Conforme estudamos, é na camada de aplicação que são executados os processos dos usuários. Nos processos em que eles interagem, realiza-se o que seus usuários esperam. Porém, para que uma aplicação possa trocar dados com outra, é necessário **definir um protocolo de aplicação**.

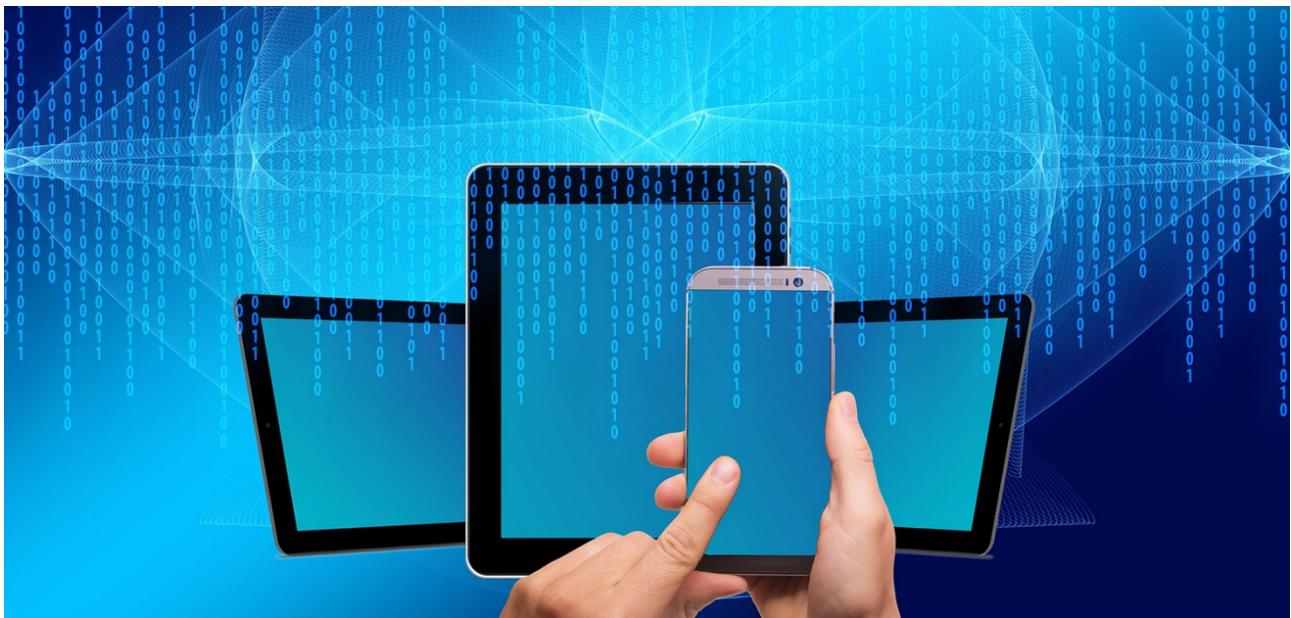
Mas o que é um protocolo da camada de aplicação?

Um protocolo de camada de aplicação define como processos de uma aplicação, que funcionam em sistemas finais diferentes, passam mensagens entre si. Em particular, um protocolo de camada de aplicação define:

- Os tipos de mensagens trocadas, por exemplo, de requisição e de resposta;
- A sintaxe dos vários tipos de mensagens, tais como os campos da mensagem e como os campos são delineados;
- A semântica dos campos, isto é, o significado da informação nos campos;
- Regras para determinar quando e como um processo envia e responde mensagens.



(KUROSE; ROSS, 2013)



Enquanto o algoritmo da camada de aplicação determina seu funcionamento no ambiente local, o protocolo dela estipula tudo que é necessário para que aplicações em diferentes hospedeiros possam trocar mensagens de maneira estruturada.

Os protocolos públicos da internet são especificados por RFCs\*. Desse modo, qualquer pessoa é capaz de acessar as especificações de tais protocolos e implementar os próprios *softwares*.

**\*RFCs:** Sigla originada do inglês *request for comments*. RFCs são documentos públicos mantidos pela *internet engineering task force* (IETF): um grupo internacional aberto cujo objetivo é identificar e propor soluções para questões relacionadas à utilização da internet, além de propor uma padronização das tecnologias e dos protocolos envolvidos.

Para que possamos compreender melhor o funcionamento das camadas de aplicação, analisaremos aquela aplicada na internet; afinal, trata-se de uma rede de abrangência mundial presente no dia a dia de milhões de pessoas.

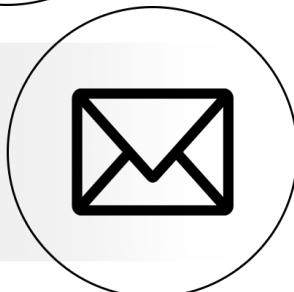
## CAMADAS DE APLICAÇÃO NA INTERNET

Descreveremos a seguir o funcionamento de **três importantes aplicações** das camadas de aplicação na internet:



### HTTP (serviço web)

Implementado pelo protocolo HTTP, que muita gente confunde com a própria internet.



### Correio eletrônico (e-mail)

Serviço de correio eletrônico.



### DNS

Sistema de resolução de nomes DNS.



## HTTP (SERVIÇO WEB)

Definido pelas RFCs **1945** e **2616**, o HTTP (*hypertext transfer protocol*) é o protocolo padrão para transferência de páginas web na internet.

Em 1991, a web foi idealizada no **CERN**\* como uma forma de fazer com que grupos de cientistas de diferentes nacionalidades pudessem colaborar por meio da troca de informações baseadas em **hipertextos\***. Em dezembro daquele ano, foi realizada uma demonstração pública na conferência **Hypertext 91\***.

**\*CERN:** Sigla originada do inglês European Organization for Nuclear Research (em português, Organização Europeia para a Pesquisa Nuclear). Trata-se de um laboratório de física de partículas localizado em Meyrin, que fica na fronteira franco-suíça.

**\*Hipertextos:** Documentos construídos com o objetivo de possuir alguns objetos, como palavras, imagens etc. Quando acionados – geralmente, com um clique do *mouse* –, eles buscam outros documentos que podem (ou não) ser hipertextos.

**\*Hypertext 91:** “A conferência sobre hipertexto e hipermídia reúne acadêmicos, pesquisadores e profissionais de diversas disciplinas para considerar a forma, o papel e o impacto do hipertexto e da hipermídia em um fórum de discussão de ideias, *design* e uso de hipertexto e hipermídia em vários domínios. A conferência também considera o poder transformador da hipermídia e sua capacidade de alterar a maneira como lemos, escrevemos, argumentamos, trabalhamos, trocamos informações e nos divertimos.”

(ACM HYPERTEXT 91 CONFERENCE, 1991, tradução nossa)

Como esse protocolo é constituído?

### Etapa 1

Uma página web típica é um documento em formato **HTML\*** que pode conter imagens e outros tipos de objetos, como vídeos, texto, som etc.

**\*HTML:** Linguagem utilizada na construção de páginas *web*. Um documento HTML possui uma série de marcadores utilizados para definir o formato a ser empregado na apresentação da página *web* ao usuário.

Para exibir determinada página *web*, o usuário digita no *browser* o endereço no qual ela se encontra (ou clica em um hiperlink para esta página), indicando o local em que deve ser buscada. Para que uma página seja transferida do servidor até o *browser*, um padrão deve ser seguido pelos *softwares* (cliente e servidor). Ele especifica como o cliente solicita a página e o servidor a transfere para o cliente.

## Etapa 2

**Esse padrão é o protocolo HTTP.** A mensagem HTTP, por sua vez, é carregada pelo por outro protocolo: **TCP\***.

**\*TCP:** Abreviação de *transmission control protocol* (TCP), trata-se do protocolo de nível de transporte confiável que garante a entrega dos dados da mensagem livre de erros no destino.

Uma interação entre cliente e servidor se inicia quando ele envia uma requisição a um servidor. A solicitação mais comum consiste em:

- Enviar um texto em formato **ASCII\***;
- Iniciar com a palavra GET;
- Inserir página solicitada, protocolo utilizado na transferência e servidor a ser contatado.

**\*ASCII:** Do inglês *american standard code for information interchange* (código padrão americano para o intercâmbio de informação), esta sigla trata de um código binário que codifica um conjunto de 128 símbolos, incluindo:

- Sinais gráficos;
- Letras do alfabeto latino;
- Sinais de pontuação;
- Sinais matemáticos;
- Sinais de controle.

## NA PRÁTICA

Vamos analisar os conceitos estudados na prática? Veja o caso a seguir.

Para solicitar a página web da Organização das Nações Unidas utilizando o protocolo HTTP, o browser estabelece uma conexão TCP com o servidor web situado no endereço [www.un.org](http://www.un.org) e lhe envia a seguinte solicitação:

Comando para solicitar página (GET)  
Arquivo a ser buscado ("/" indica página iniciada)  
Utilizar protocolo HTTP  
Versão do protocolo HTTP (1.1)

**GET / HTTP/1.1**

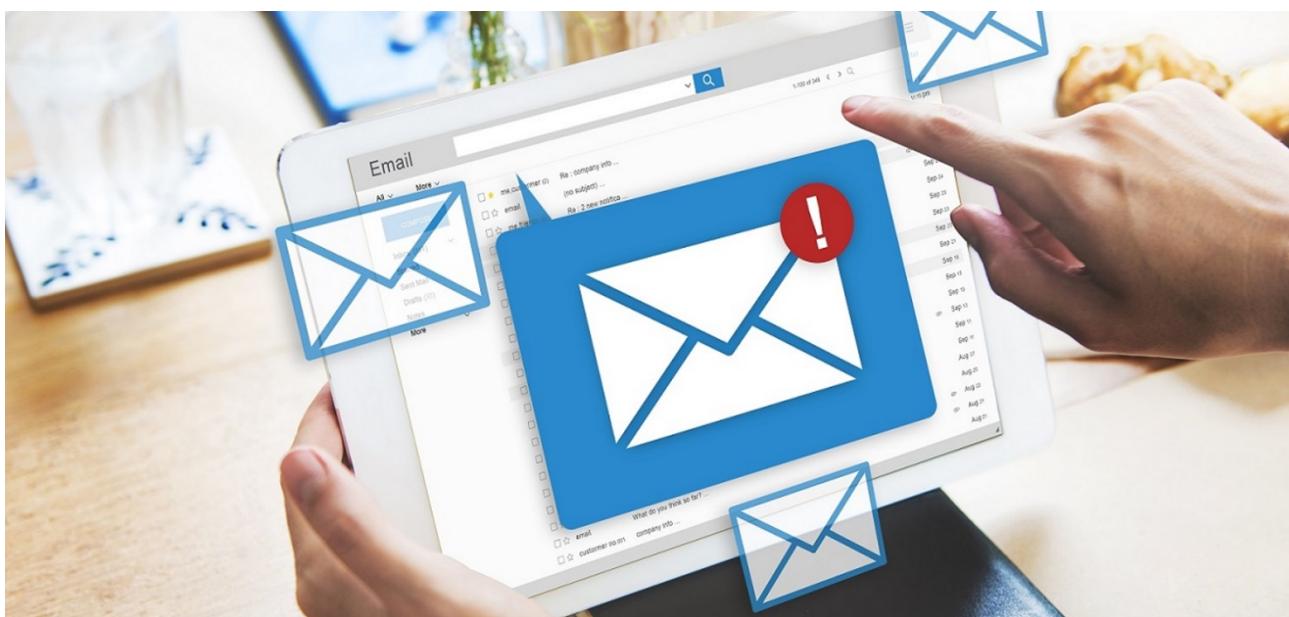
**Host: www.un.org**

Acessar arquivo no servidor “www.un.org”

### Como esse processo é organizado?

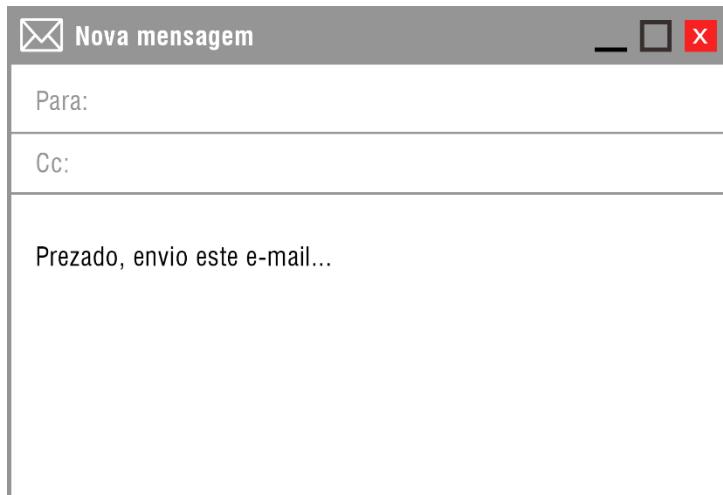
Ao receber a solicitação, o servidor busca a página *web* solicitada, a transfere para o cliente e, após confirmada a entrega, encerra a conexão.

Como o HTTP utiliza o TCP, não é necessário se preocupar com questões de confiabilidade na entrega dos dados. Ele é um protocolo em constante evolução, havendo atualmente várias versões em uso. Por isso, o cliente deve informar a versão do protocolo a ser usado quando solicita uma página web.



# CORREIO ELETRÔNICO (E-MAIL)

Trouxemos um exemplo para esclarecer essa questão:



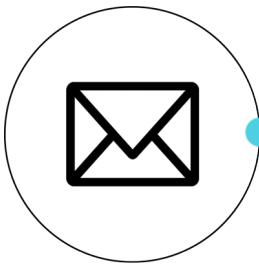
Os primeiros sistemas de correio eletrônico foram concebidos como um simples sistema voltado para a troca de arquivos. O destinatário da mensagem era especificado na primeira linha do texto.

Bastava então que o sistema procurasse ali para quem a mensagem deveria ser entregue. Porém, com o passar do tempo, surgiram novas necessidades que dificilmente eram atendidas por ele.

Em 1982, ainda na era da ARPANET<sup>\*</sup>, foram publicadas as RFCs **821** e **822**, definindo, respectivamente, o protocolo de transmissão a ser utilizado e o formato da mensagem. Entretanto, apesar de ambas resolverem o problema inicial a que se propunham, elas especificavam que todo o texto deveria ser composto pelo código ASCII.

**\*ARPANET:** Precursora da internet, ela foi a primeira rede a implementar o conjunto de protocolos TCP/IP.

Tal restrição precisava ser resolvida para ser possível o **envio de mensagens**:



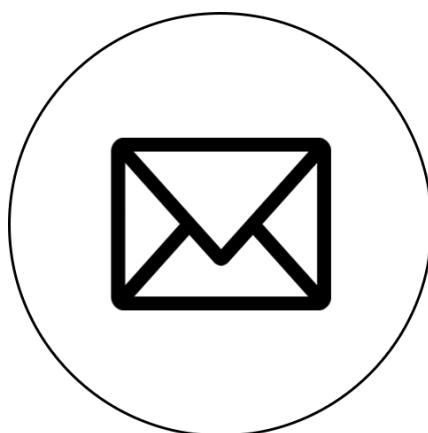
- Em alfabetos não latinos.
- Em idiomas sem alfabetos.
- Que não contêm textos multimídia, como, por exemplo, áudio e vídeo.
- Com caracteres acentuados.



Para resolver esses novos problemas, foi criada uma **solução** denominada ***multipurpose internet mail extensions (MIME)***. O MIME continua utilizando o formato da RFC 822, mas passou a incluir uma estrutura para o corpo da mensagem e **definir regras para as mensagens especiais**.

Essa estratégia fez com que tais mensagens pudessem ser enviadas graças à utilização de protocolos e programas de correio eletrônico existentes, existindo somente a necessidade de alterar os programas de envio e recebimento.

Atualmente, o protocolo de transmissão *simple mail transfer protocol (SMTP)* é definido pela **RFC 5321**, enquanto o formato da mensagem o é pela **RFC 5322**.



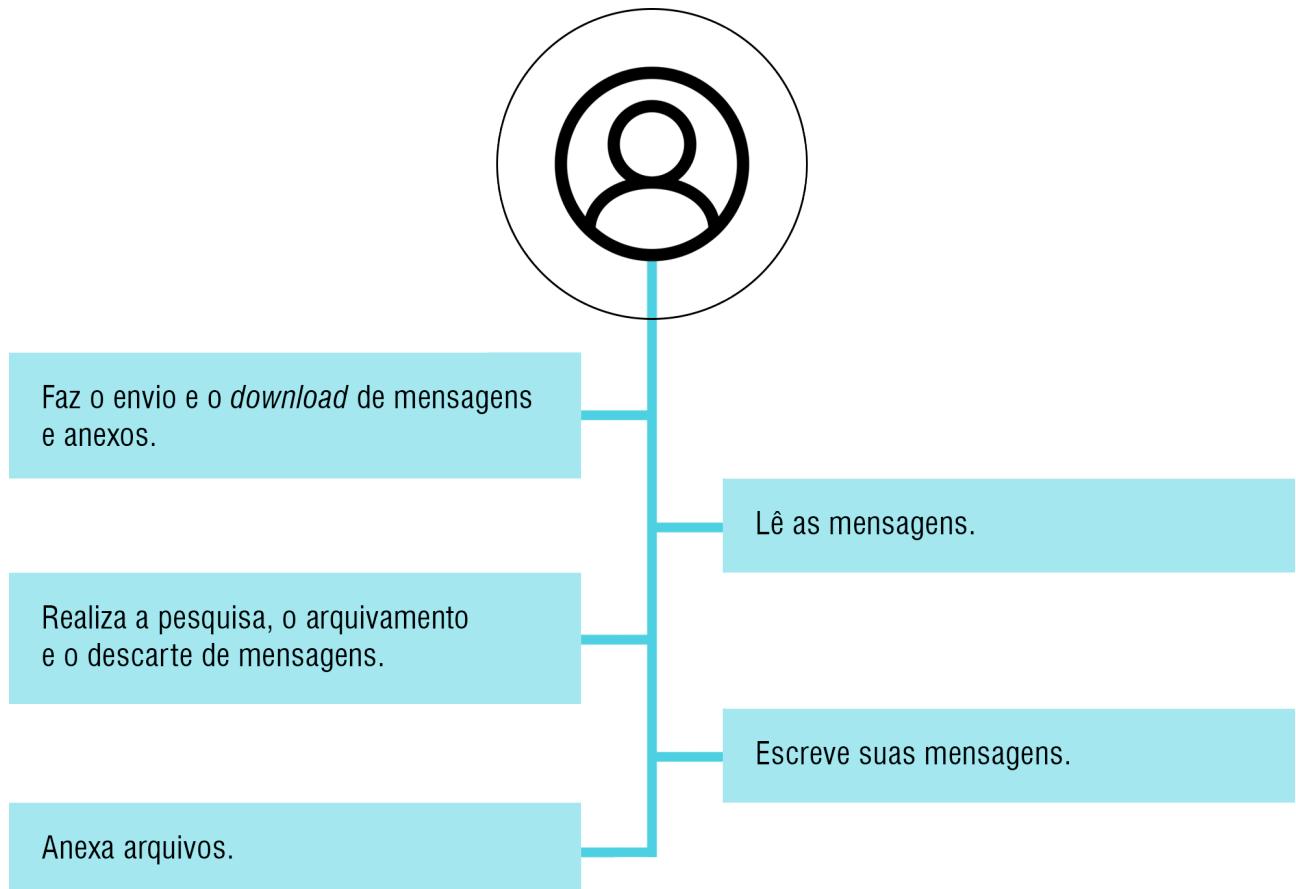
Como é construída a arquitetura do correio eletrônico?

A arquitetura do sistema de correio eletrônico é construída com base em **dois agentes**:

1. Do usuário;
2. De transferência de mensagens.

O **agente do usuário** é o programa que faz a interface do usuário com o sistema de correio eletrônico.

É por meio dele que o **usuário**:



Mostraremos a seguir alguns desses programas:



Mozilla  
Thunderbird



Microsoft  
Outlook



Eudora

**Já os agentes de transferência de mensagens** são os responsáveis por fazer com que elas cheguem até o destino. Eles são mais conhecidos como **servidores de correio eletrônico**.



Postfix

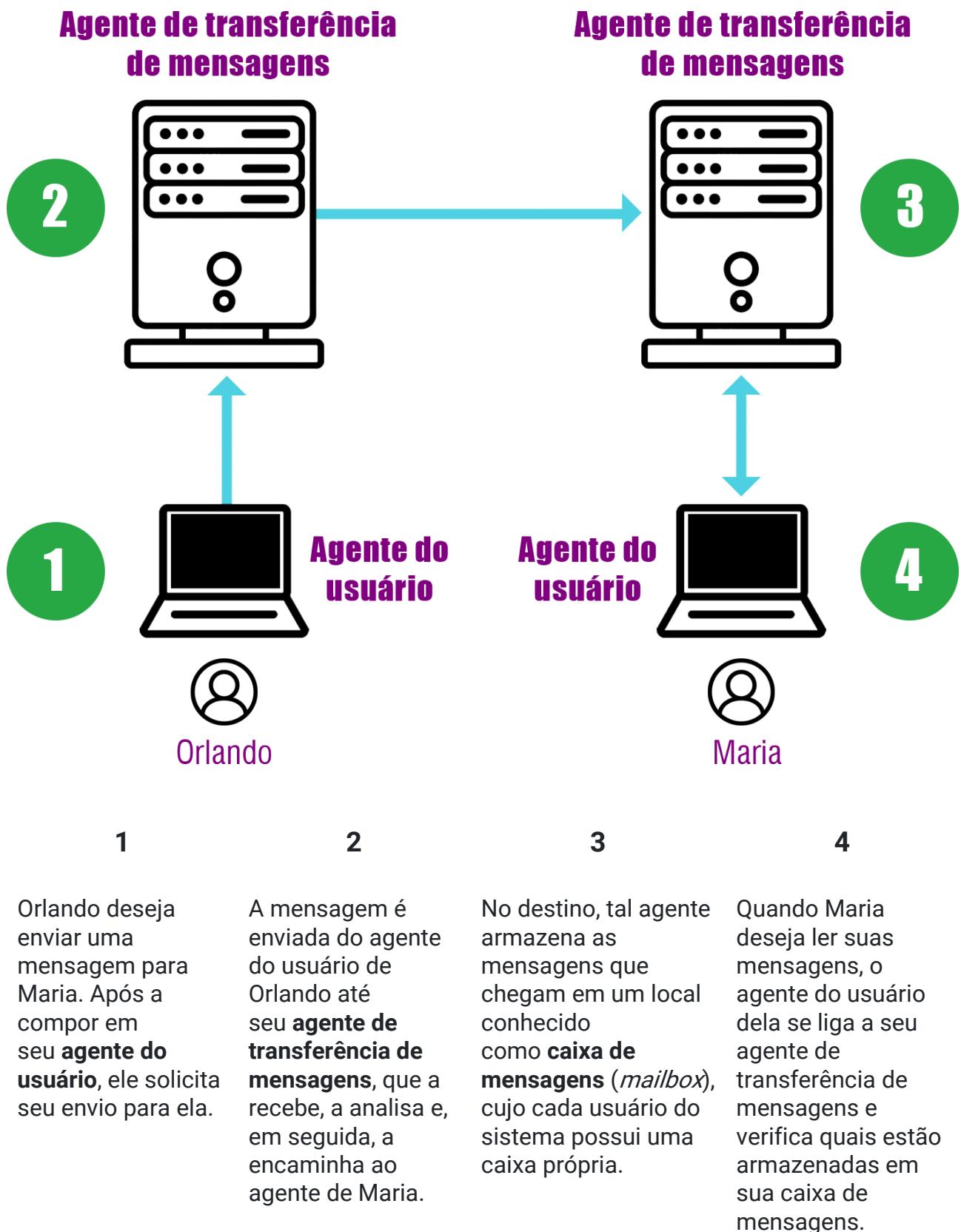


Zimbra



Exchange

Para entendermos melhor o assunto, analisaremos a seguir a comunicação entre Orlando e Maria. Esse caso explicita uma arquitetura do sistema de correio eletrônico:



Para concluirmos esse estudo, analisaremos importantes características dos protocolos apresentados:

## SMTP

O protocolo responsável pela transferência da mensagem até seu destino é o SMTP. Definido pela RFC 5321, **ele utiliza o protocolo de transporte TCP, obtendo, assim, a garantia de que ela será entregue no destino sem erros.**

O servidor SMTP aguarda por conexões de seus clientes. Quando uma conexão é estabelecida, o servidor inicia a conversação enviando uma linha de texto na qual se identifica e informa se está pronto (ou não) para receber mensagens. Se ele não estiver, o cliente deverá encerrar a conexão e tentar novamente mais tarde.

Caso o servidor esteja acessível, o cliente precisa informar aos usuários a origem e o destino da mensagem. Se o servidor considerar que se trata de uma transferência válida, sinalizará para que ele a envie. Após o envio, o servidor confirma sua recepção e a conexão é encerrada.

### Exemplo:

Retomando o caso da comunicação anterior, podemos ver, na sequência apresentada adiante, a conversação entre cliente e servidor para estabelecer a transferência da mensagem de orlando@origem.net para maria@destino.net:

```
220 Protegido* SMTP server
hello rayra.origem.net
250 Hello rayra.origem.net, pleased to meet you
mail from:
250 ... Sender ok
rcpt to: < maria@destino.net >
250 < maria@destino.net >... Recipient ok
data
354 Please start mail input.
subject: Teste de email
Primeira linha da mensagem de teste.
Segunda linha.
Quarta linha.
.
250 Mail queued for delivery.
```

**\*Protegido:** “Protegido” é a identificação do servidor que recebe a mensagem; “rayra.origem.net”, o nome do hospedeiro que a envia.

## Entrega final

Quando uma mensagem chega ao servidor do destinatário, ela deve ser armazenada em algum local para que possa ser acessada mais tarde (assim que o destinatário estiver *on-line*). Este local é a caixa de mensagens.

Como o SMTP é responsável somente pela entrega da mensagem no servidor destino, isso requer a utilização de outro protocolo de modo que o cliente possa buscar suas mensagens no *mailbox*.

### POP3

A **RFC 1939** estipula que o POP3 (*post office protocol version 3*) tem a **finalidade de fazer o download das mensagens que se encontram no *mailbox* do usuário para o sistema local**. Caso estejam neste sistema, ele pode utilizá-las em qualquer momento, mesmo sem ter conexão com a internet.

O POP3 é implementado na maioria dos agentes de usuário. Basta configurar os parâmetros de conta e senha do usuário para que o agente faça o *download* das mensagens. Ele permite o *download* seletivo delas, assim como apagar as selecionadas no servidor.

### IMAP

Assim como o POP3, o IMAP (*internet message access protocol*) permite que um usuário tenha acesso às mensagens armazenadas em sua caixa. Porém, enquanto o POP3 é baseado na transferência delas para o sistema local a fim de serem lidas, o IMAP consegue permitir sua leitura diretamente no servidor, dispensando, portanto, a transferência para o sistema local.

Isso será particularmente útil para usuários que não utilizarem sempre o mesmo computador, pois isso permite que suas mensagens possam ser acessadas a partir de qualquer sistema. Definido pela **RFC 3501**, o IMAP também fornece mecanismos para criar, excluir e manipular várias caixas de correio no servidor.

## Atenção!



Um *webmail* não é um protocolo, mas **uma forma oferecida por alguns sites da web a fim de que os usuários possam ler suas mensagens de correio eletrônico.**

Para usar o sistema, o usuário abre uma página *web*, na qual entra com uma identificação e uma senha. A partir desse momento, ele tem acesso imediato às suas mensagens (de forma parecida com a de um cliente IMAP).



## DNS

A comunicação entre hospedeiros na internet ocorre por meio de **endereços binários de rede**. Afinal, para se comunicar com um destino, o hospedeiro precisa conhecer seu endereço.

Entretanto, é bem mais fácil trabalhar com nomes de hospedeiros do que com seus endereços de rede. Além de ser muito difícil conhecer todos os endereços dos hospedeiros com os quais precisamos trabalhar, precisaríamos ser notificados toda vez que algum deles mudasse de endereço.

Para resolver esse problema, foi desenvolvido o *domain name system* (DNS). Sua finalidade é a criação de um sistema de nomes de forma hierárquica e baseada em

domínios. Para acessar um hospedeiro, portanto, basta conhecer seu nome de domínio e fazer uma consulta ao servidor DNS, que é responsável por descobrir seu endereço.



## Quais são os serviços oferecidos por ele?

Além do **mapeamento de nomes de hospedeiros em endereços IP**, o DNS ainda provê:

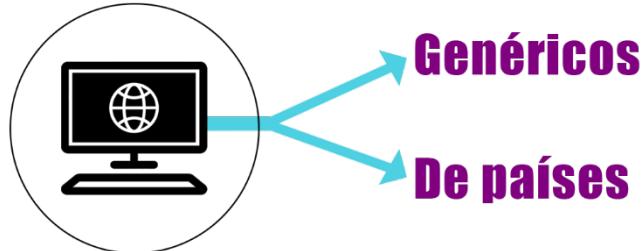
- Identificação de servidores de correios eletrônicos;
- Apelidos para hospedeiros;
- Distribuição de carga;
- Descoberta de nomes de hospedeiros (mapeamento reverso).

Destacaremos a seguir importantes aspectos do DNS.

### Espaço de nomes

O espaço de nomes do DNS é dividido em **domínios estruturados em níveis**. Confira a organização do **primeiro nível**:

#### Domínios

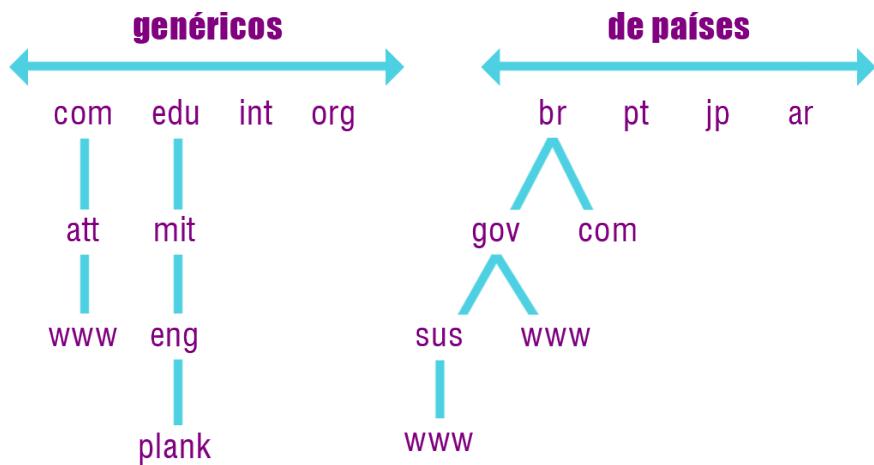


Os **domínios genéricos** informam o tipo de organização ao qual o domínio está vinculado. Alguns exemplos são:

- *.com* = comercial;
- *.edu* = instituições educacionais;
- *.int* = algumas organizações internacionais;
- *.org* = organizações sem fins lucrativos.

Os **domínios de países**, por sua vez, possuem uma entrada para cada país. Alguns exemplos são:

- *.br* = Brasil;
- *.pt* = Portugal;
- *.jp* = Japão;
- *.ar* = Argentina.



Cada domínio tem seu nome definido pelo caminho entre ele e a raiz, enquanto seus componentes são separados por pontos.

Cada domínio controla como são criados seus subdomínios. Para a criação de um novo domínio, é necessária apenas a permissão daquele no qual será incluído.

Não há qualquer restrição sobre a quantidade de subdomínios que podem ser criados dentro de um domínio. Os nomes de domínio não fazem distinção entre letras maiúsculas e minúsculas.

**EDU** e **edu**, por exemplo, são o mesmo.

Os nomes de componentes podem ter até 63 caracteres, enquanto os de caminhos completos não podem ultrapassar os 255.

O DNS é implementado sobre o protocolo UDP (*user datagram protocol*). Trata-se de um protocolo do nível de transporte que não garante a entrega dos dados no destino. Dessa forma, cabe ao software DNS garantir uma comunicação confiável.

## Resolução de nomes

O espaço de nomes do DNS é dividido em **zonas**. Independentes, elas possuem um servidor de nomes principal e pelo menos um de nomes secundário:

- **Servidor de nomes principal:** Configurado com as informações das zonas sob sua responsabilidade, ele faz o repasse delas para os servidores de nome secundários;
- **Servidor de nomes secundário:** Responde pelas zonas caso haja uma falha do servidor de nomes principal.

As zonas do DNS definem o que um servidor deve resolver. Se ele for o responsável pela zona pesquisada (servidor autoritativo), deverá fazer a resolução solicitada.

### 💡 Três principais componentes do DNS:

1. Registros de recursos armazenados em um banco de dados distribuído;
2. Servidores de nomes DNS responsáveis pela manutenção de zonas específicas;
3. Solucionadores DNS em execução nos clientes.

### 💡 Solucionador X servidor DNS.

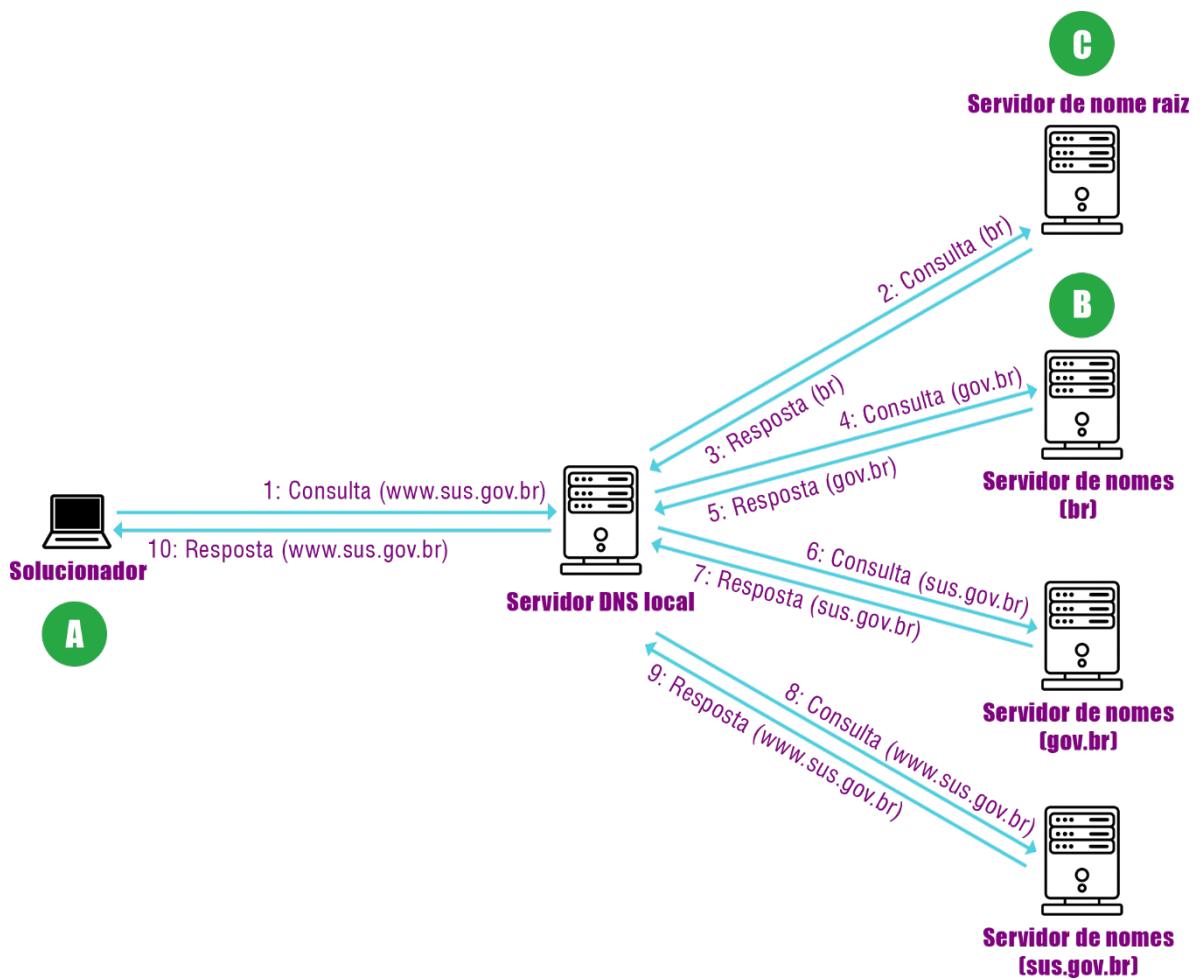
Quando um solucionador solicita a resolução de um nome para o servidor DNS, pode acontecer o seguinte:

**1. O servidor DNS é o responsável pela zona:** O servidor resolve o nome solicitado e o devolve ao solucionador;

**2. O servidor DNS não é o responsável pela zona, mas possui a resolução em cache:** O servidor envia a resolução ao solucionador;

**3. O servidor DNS não é o responsável pela zona nem possui a resolução em cache:** O servidor precisa realizar uma busca para resolver o nome.

Vamos entender como é feita a busca para a resolução do nome [www.sus.gov.br](http://www.sus.gov.br):



**A:** Quando a aplicação do cliente solicita a resolução do nome `www.sus.gov.br`, o **solucionador** envia a requisição para o servidor de nomes local, que é o responsável por tratá-la até obter a resposta completa. Desse modo, ele não retorna respostas parciais para o solucionador. A este tipo de consulta damos o nome de consulta recursiva.

**B:** No entanto, para obter a resposta completa, o **servidor de nomes** precisa realizar uma série de iterações com outros servidores. Caso nenhuma informação parcial esteja em seu *cache*, o servidor local primeiramente precisa descobrir quem é o servidor responsável por resolver o domínio `br`.

**C:** Para isso, ele consulta um servidor de nomes raiz, que indica onde o servidor DNS de “`br`” pode ser encontrado. O servidor local continua realizando consultas para resolver cada domínio parcial até que haja uma resolução completa. Este tipo de consulta é conhecido como **consulta iterativa**.

O excesso de consultas em um servidor DNS pode levar à **sobrecarga**.

#### Como evitar esse tipo de problema?

Os servidores devem evitar responder consultas recursivas de clientes não autorizados. Para isso, os administradores de servidores DNS precisam configurar no servidor aqueles autorizados a realizar consultas recursivas. Dessa forma, se houver a consulta de um que não esteja, ela automaticamente será negada.

## VERIFICANDO O APRENDIZADO

1. Cabe ao protocolo da camada de aplicação definir como funcionam os processos de uma aplicação. Nesse sentido, **não** é função dele definir:

- a) Os tipos de mensagens trocadas.
- b) O significado de cada campo do protocolo.
- c) Quando um processo pode enviar e responder mensagens.
- d) O endereço de rede do servidor.

### Comentário

**Parabéns!** A alternativa **D** está correta.

Servidores de aplicações podem estar espalhados por todo o mundo, cada um deles com o próprio endereço. Dessa forma, o protocolo da camada de aplicação não pode determinar em qual endereço de rede o servidor está localizado.

2. Um sistema de correio eletrônico é um exemplo de sistema implementado com base em uma série de protocolos. Dentre os protocolos a seguir, selecione o único que **não** é utilizado pelos sistemas de correio eletrônico:

- a) HTTP
- b) SMTP
- c) POP3
- d) IMAP

### Comentário

**Parabéns!** A alternativa **A** está correta.

O HTTP é o protocolo utilizado para transferência de páginas web. Ele não é usado como protocolo de correio eletrônico.

## Módulo 3

Localizar os elementos da camada de transporte



## CAMADA DE TRANSPORTE

Graças a essa abordagem, podemos compreender os conceitos do protocolo TPC/IP, bem como os exemplos práticos aplicados na camada de transporte. Além disso, percebemos a importância da interface para esta camada.

### Mas para que serve a camada de transporte?

Executadas na camada de aplicação, as aplicações precisam de um modelo de rede no qual haja a entrega de uma mensagem (ou um fluxo de dados) tanto em um ponto de rede quanto em sua aplicação par no hospedeiro destino.

O objetivo da camada de transporte é, independentemente das redes físicas em uso, promover a confiabilidade na transferência de dados entre os hospedeiros origem e destino.

Como veremos no decorrer do nosso estudo, esta camada deve oferecer um **serviço de transferência confiável**, embora caiba à aplicação decidir sobre o seu uso.

## SERVIÇO DE TRANSPORTE

Em uma arquitetura de camadas, podemos afirmar que o objetivo geral de uma camada é oferecer serviços àquela imediatamente superior. No caso da camada de transporte, sua pretensão é oferecê-los à de aplicação.



## Atenção!

Lembre-se de que, em nosso estudo, estamos considerando a arquitetura TCP/IP, na qual não existem as camadas de sessão e de apresentação.

Como um dos principais objetivos da camada de transporte é oferecer um serviço confiável e eficiente a seus usuários, ela precisa oferecer, no mínimo, um serviço orientado à conexão e outro **sem conexão**.

Para atingir esse objetivo, a camada de transporte utiliza os serviços oferecidos pela rede. No **serviço de transporte orientado à conexão** (serviço confiável), existem três fases:



Por meio de um controle apurado da conexão, esse serviço de transporte consegue verificar quais pacotes chegaram com erro ao destino e até mesmo aqueles que não foram enviados, sendo capaz de retransmiti-los até que os dados estejam corretos.

Já no **serviço de transporte sem conexão**, não existe nenhum controle sobre os pacotes enviados. Se um deles se perder ou chegar ao destino com erro, nada será feito para obter a sua recuperação.

Se a rede oferece um serviço com que garanta uma entrega sem erros, por que uma aplicação optaria por um serviço sem essa garantia?

A resposta é simples: **por questões de desempenho**.

Pelo fato de ser preciso cuidar de cada pacote no serviço orientado à conexão, verificando-os e retransmitindo-os em caso de necessidade, esse controle gera um **overhead**\*. Como nada disso é feito no serviço sem conexão, os pacotes são entregues no destino de forma mais simples e rápida.

**\*Overhead:** Termo em inglês utilizado com frequência em computação para indicar uma sobrecarga no sistema. No caso do serviço orientado à conexão, o *overhead* ocorre graças ao processamento extra necessário para a verificação dos dados e uma eventual retransmissão.



Aplicações como transferência de arquivos e e-mail exigem que seus dados cheguem ao destino livres de erros. Dessa forma, elas utilizam um serviço orientado à conexão.

Ainda assim, em certas aplicações, o mais importante é a **chegada a tempo de uma informação, mesmo que ela contenha erros ou que a mensagem anterior tenha se perdido.**

No serviço de telefonia em rede, por exemplo, o atraso na transmissão tem um efeito pior que um pequeno ruído causado pela eventual perda de pacote.

## Endereçamento

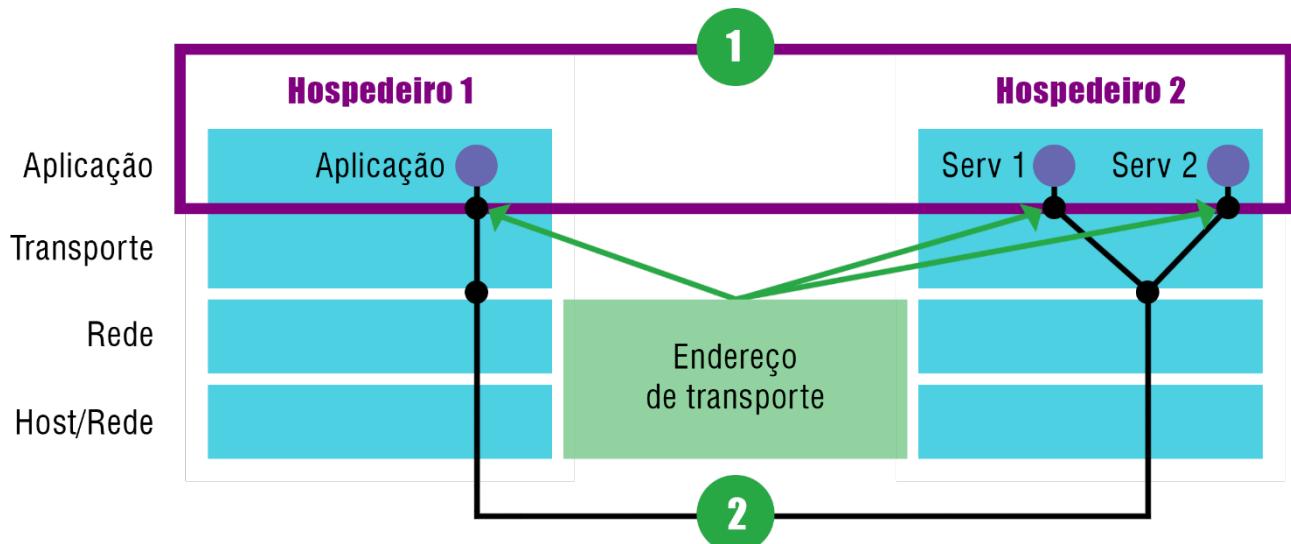
Quando seu programa solicita algo a um servidor, o sistema envia uma mensagem para ser entregue à aplicação que executa em um hospedeiro remoto. Mas podem existir várias aplicações nele.

### Como identificamos uma aplicação específica?

Surge neste momento o endereçamento no nível de transporte. Sua função é identificar em qual aplicação determinada mensagem deve ser entregue. Afinal, toda mensagem do protocolo de transporte carrega o endereço da aplicação.

Verificaremos agora a importância do endereçamento no nível de transporte. Afinal, é necessário indicar em qual aplicação os dados devem ser entregues por meio de seu endereço (de transporte). Assim, o hospedeiro destino consegue saber o destino deles.

Estudaremos mais adiante TCP e UDP, dois protocolos da camada de transporte da arquitetura TCP/IP. Neles, o endereço de transporte é conhecido como **porta**.



Como a aplicação do **hospedeiro 1** sabe em que endereço de transporte se encontra o **servidor no 2**? Uma possibilidade é que:

- Ele esteja associado ao endereço há anos;
- Aos poucos, todos os usuários da rede tenham se acostumado com isso.

Neste modelo, os serviços possuem **endereços estáveis** que podem ser impressos e distribuídos aos novos usuários quando eles se associam à rede.



### Servidor de nomes ou de diretórios

Um esquema alternativo é utilizar um processo especial denominado **servidor de nomes** (*name server*) ou, às vezes, **servidor de diretórios** (*directory server*). Para localizar o endereço de transporte correspondente a determinado nome de serviço, uma aplicação estabelece uma conexão com o servidor de nomes. Em seguida, envia uma mensagem especificando o nome do serviço, enquanto o servidor de nomes retorna o endereço.

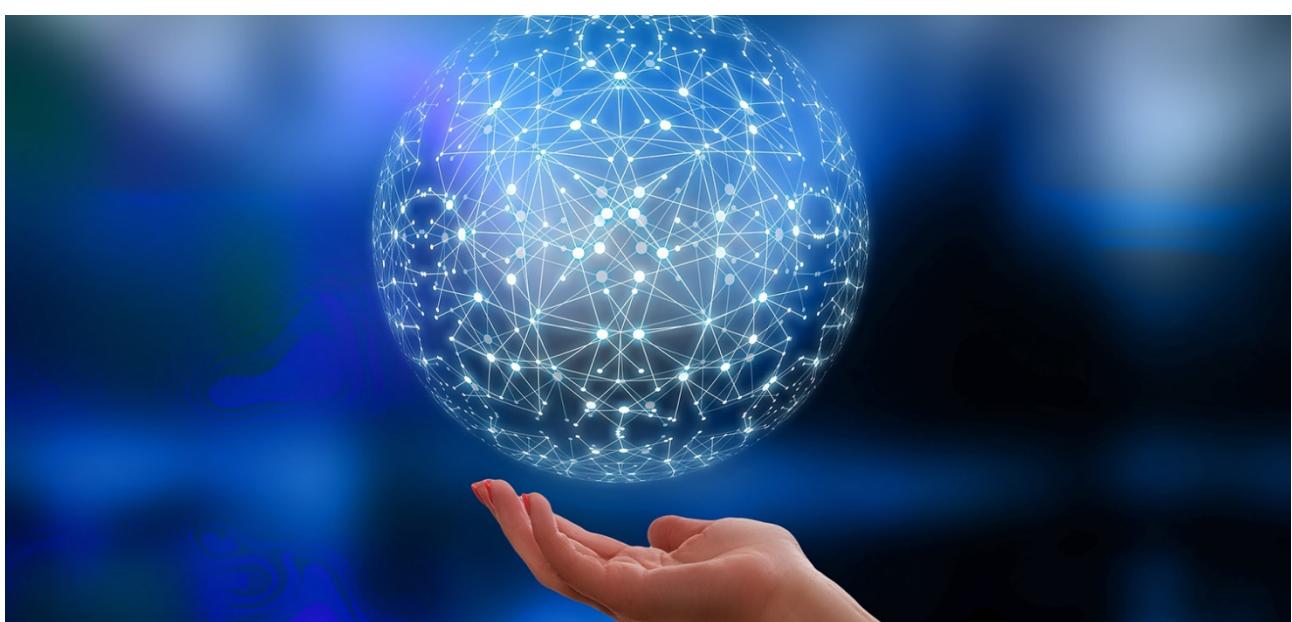
## Multiplexação e demultiplexação

A multiplexação e a demultiplexação fornecem um serviço de entrega processo a processo para aplicações executadas nos hospedeiros.

No hospedeiro destino, a camada de transporte recebe **segmentos\*** de dados da camada de rede, tendo a responsabilidade de entregá-los ao processo de aplicação correto.

**\*Segmentos:** Cada camada do modelo de rede denomina os dados trocados com o hospedeiro remoto de uma forma diferente das demais camadas. **Segmento** é o nome da mensagem trocada entre duas entidades de transporte tanto no modelo OSI quanto na arquitetura TCP/IP.

Exemplo: a camada de rede chama suas mensagens de pacotes. Já a de enlace de dados as nomeia como **quadros**.



Um processo pode ter um ou mais endereços de transporte (conhecidos como portas na arquitetura TCP/IP) pelos quais dados passam da rede para o processo – e vice-versa.

Desse modo, a camada de transporte do hospedeiro destino os entrega diretamente a uma porta.

### Como o hospedeiro destino direciona à porta correta um segmento que chega?

Para essa finalidade, cada segmento da camada de transporte tem um conjunto de campos de endereçamento no cabeçalho. No receptor, a camada de transporte examina

esses campos para identificar a porta receptora e direcionar o segmento a ela. A tarefa de entregar os dados contidos em um segmento para a porta correta é denominada **demultiplexação**.



Já a **multiplexação** consiste no trabalho de, no hospedeiro origem:

- Reunir porções de dados provenientes de diferentes portas;
- Encapsular cada porção de dados com as informações de cabeçalho (as quais, mais tarde, serão usadas na demultiplexação) para criar segmentos;
- Passar os segmentos para a camada de rede.

**Vamos pensar no computador que Eduardo utiliza em suas atividades.**



Navegando na web, ele acessa seu e-mail e faz o *download* de arquivos usando um programa específico para isso.

De fato, o objetivo da multiplexação é possibilitar uma **melhor utilização do meio de comunicação** ao permitir que ele seja compartilhado pelos diversos programas utilizados.



Eduardo utiliza a internet, cujo protocolo de transporte é o **TCP**.



Todos os programas operados por ele (*browser web*, cliente de e-mail e programa de transferência de arquivos) utilizam o TCP, que fará a transferência da informação até o destino.

A **multiplexação**, portanto, permite que vários programas possam utilizar o TCP ao mesmo tempo, fazendo, assim, com que Eduardo possa ter tantos programas quanto queira ao acessar a rede.

## Como o TCP sabe quem é quem?

Para fazer uso dele, um processo deve se registrar em uma porta (endereço de transporte) do protocolo TCP. Servidores possuem portas conhecidas, mas programas clientes se registram nas aleatórias.

Vamos supor que os programas de Eduardo se registraram nas seguintes portas:

**Browser web = 11278**  
**Cliente de e-mail = 25786**  
**Transferência de arquivos = 3709**

Dessa maneira, o TCP pode identificar cada uma. Quando o *browser* envia uma solicitação a um servidor *web*, o TCP coloca na informação enviada o número de porta 11278. O servidor, portanto, já sabe que deve responder-lhe enviando a resposta para esta porta.

Observemos, por fim, a multiplexação e a demultiplexação na prática:



## VERIFICANDO O APRENDIZADO

1. Para qual das aplicações a seguir é recomendado o uso de um serviço de transporte sem conexão?

- a) Telefonia
- b) Correio eletrônico
- c) Serviço *web*
- d) Transferência de arquivos

### Comentário

**Parabéns!** A alternativa A está correta.

No serviço de telefonia, uma eventual tentativa de correção de erros provocaria atraso na chegada da fala de uma pessoa em seu destino. Atrasos são muito prejudiciais neste tipo de serviço, pois os interlocutores se confundem quando ocorrem defasagens entre a transmissão e a recepção da mensagem.

2. O endereçamento no nível de transporte é importante para a realização de:

a) Identificação do hospedeiro

b) Correção de erros

c) Encerramento da conexão

d) Demultiplexação

### Comentário

**Parabéns!** A alternativa D está correta.

Como a camada de transporte precisa identificar a qual dos processos da camada de aplicação ela deve entregar os dados que chegam, são definidos campos de endereçamento de transporte para a identificação desses processos. Dá-se o nome de demultiplexação à identificação e ao envio de informações para o processo correto.



# Comparar os serviços oferecidos pela camada de transporte

## PROTOCOLOS DE TRANSPORTE DA INTERNET

Agora que já estudamos os serviços que um protocolo de transporte deve oferecer, apresentaremos casos reais de protocolos utilizados em redes de computadores. Para isso, vamos utilizar como exemplo os protocolos de transporte da internet: **TCP e UDP**.

Iniciaremos nosso estudo pelo **UDP**. Mesmo sendo um protocolo simples, ele se revela bastante eficiente, principalmente no quesito agilidade de entrega, quando a aplicação requer uma entrega rápida.

Em seguida, nos debruçaremos sobre o TCP. Protocolo de transporte completo, ele é capaz de garantir a entrega de mensagens livres de erros, não importando a qualidade da rede em que ele trabalha.



Por fim, apontaremos alguns exemplos de portas, um mecanismo que permite que as aplicações serem encontradas pela internet.

## UDP

O protocolo de transporte mais simples que poderia existir seria aquele que, no envio, se limitasse a receber mensagens da camada de aplicação e as entregasse diretamente na rede. Na recepção, ele, por outro lado, receberia os pacotes da camada de rede e os entregaria na de aplicação. **Este tipo de protocolo, em suma, não efetua nenhum trabalho para garantir a entrega das mensagens.** Felizmente, o UDP não se limita a isso.



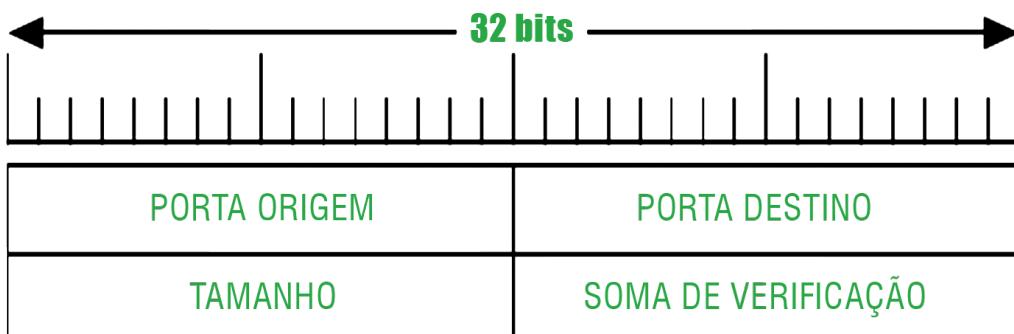
## Atenção!

O UDP é um protocolo rápido na entrega das mensagens no destino, realizando a multiplexação e demultiplexação; além disso, ele oferece um mecanismo de verificação de erros nessa entrega.

Um processo pode ter um ou mais endereços de transporte (conhecidos como portas na arquitetura TCP/IP) pelos quais dados passam da rede para o processo – e vice-versa.

Desse modo, a camada de transporte do hospedeiro destino os entrega diretamente a uma porta.

A figura a seguir ilustra os campos do cabeçalho de um segmento UDP:



Os campos **porta origem** e **porta destino** têm a função de identificar os processos nas máquinas origem e destino. Quando uma aplicação **A** deseja enviar dados para uma **B**, o UDP coloca o número da porta da aplicação origem (**A**) no campo “porta origem” e o de porta da aplicação (**B**) em “porta destino”.

Quando a mensagem chega ao destino, o UDP pode entregá-la para a aplicação correta por meio do campo “porta destino”. Já a “porta origem” é importante para a aplicação que recebe a mensagem, pois ela torna possível saber o número da porta para a qual a resposta deve ser enviada. É por meio desses campos que o UDP realiza a multiplexação e a demultiplexação.

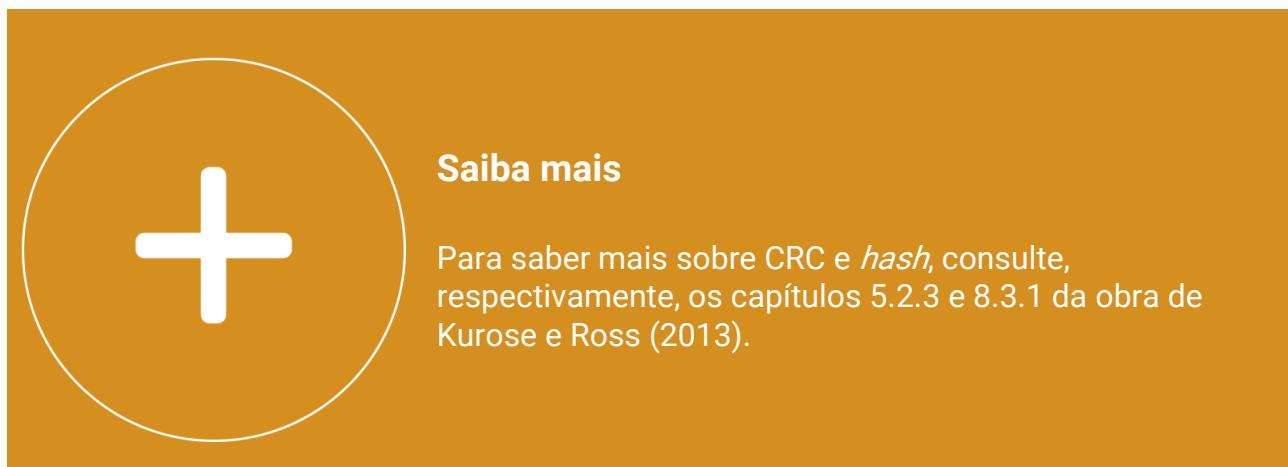
O campo **tamanho** especifica o tamanho do segmento, incluindo o cabeçalho. Como se trata de um campo de 16 *bits*, isso significa que o maior segmento UDP será de:

$$2^{16} = 65.536 \text{ bytes (64 KBytes)}$$

O campo **soma de verificação** tem a função de garantir que a mensagem chegue ao destino livre de erros. Para tanto, o UDP calcula o CRC\* dela e o envia neste campo. No destino, o CRC é novamente calculado e comparado. Se ambos forem iguais, a mensagem é considerada livre de erros e entregue na aplicação destino.

**\*CRC:** Sigla que vem do inglês *cyclic redundancy check*, ela constitui um método de detecção de erros utilizado por sistemas computacionais. Calcula-se nele o *hash* de uma mensagem a ser anexado a seu final. No destino, um novo *hash* é calculado e comparado com o recebido. Se eles forem iguais, assume-se que a transmissão ocorreu sem erros.

O *hash* é uma função matemática muito utilizada na criptografia. Ao fazer o processamento de dados de tamanho arbitrário, ele sempre gera uma saída de tamanho fixo para comparação.



Caso haja alguma divergência no valor do CRC, o segmento normalmente é descartado, porém algumas implementações permitem – acompanhadas de uma mensagem de aviso – a entrega dele com erro.

O UDP é um **protocolo sem estado\*** e não orientado à conexão. Descrito pela **RFC 768**, ele é projetado tanto para pequenas transmissões de dados quanto para aqueles que não requerem um mecanismo de transporte confiável.

Apesar de o UDP não oferecer uma confiabilidade nas transmissões, isso não significa que aplicações que o utilizam não possam ter uma garantia de entrega.

**\*Protocolo sem estado:** Protocolo que não leva em consideração o que foi realizado no passado. Ele, portanto, trata toda requisição como algo isolado, sem qualquer relação com requisições passadas ou futuras.



## Atenção!

Cabe ao programador cuidar dessa garantia no código da própria aplicação.



Protocolos de aplicações que utilizam o UDP:

**DNS**

**SNMP**

**TFTP**

**RPC**

## TCP

Enquanto o UDP é um protocolo de transporte simples, voltado para aplicações que não necessitam de confiabilidade na transmissão, o TCP é um **orientado à conexão**, sendo indicado para aplicações que precisam trocar uma grande quantidade de dados por meio de uma rede com múltiplos roteadores.



O TCP oferece um fluxo de *bytes* fim a fim confiável, podendo ser utilizado, inclusive, em redes de baixa confiabilidade. Na transmissão, ele aceita fluxos de dados da aplicação: dividindo-os em partes de, no máximo, 64 KBytes, ele envia cada uma em um **datagrama IP\*** distinto. Quando os datagramas IP com dados TCP chegam ao hospedeiro destino, eles são, em seguida, enviados à entidade TCP, que restaura o fluxo de dados original.

**\*Datagrama IP:** Também conhecido como “pacote”, um datagrama é uma porção de dados trocada por protocolos da camada de rede. Ele contém dados e informações de cabeçalho suficientes para que eles possam seguir seu caminho até o hospedeiro destino.

A camada de rede (protocolo IP) não oferece nenhuma garantia de que os datagramas serão entregues corretamente. Portanto, cabe ao TCP administrar os temporizadores e retransmitir os datagramas sempre que for necessário.

Os datagramas também podem chegar fora de ordem, cabendo a ele reorganizá-los em mensagens na sequência correta. **O TCP deve fornecer a confiabilidade que o IP não oferece.**

O TCP é definido pelas seguintes RFCs:

**793 2018 1122 2581 1323**

Para concluirmos nossa última etapa de estudos, precisamos entender **três aspectos fundamentais da TCP**:



Modelo de serviço TCP



Cabeçalho de segmento TCP



Gerenciamento de conexão TCP

## Modelo de serviço TCP

O serviço TCP é obtido quando tanto o transmissor quanto o receptor criam pontos terminais; denominados **portas**, eles são identificados por um número de 16 *bits*. É necessário que uma conexão seja explicitamente estabelecida entre um hospedeiro transmissor e um receptor.

Todas as conexões TCP são:

### Full-duplex

Dados podem ser enviados e recebidos por ela simultaneamente. Uma linha telefônica é um exemplo de sistema *full-duplex*, pois permite que dois interlocutores falem de forma simultânea. O *walkie-talkie*, no entanto, é diferente: como tal equipamento está no modo de transmissão ou no de recepção, ele nunca consegue transmitir e receber ao mesmo tempo.



### Ponto a ponto

Interliga diretamente dois hospedeiros, não permitindo a participação de um terceiro na conversação. Exemplo: quando ligamos um *smartphone* a um computador por meio de seu cabo de dados, ocorre uma ligação ponto a ponto, pois somente eles podem utilizar esse meio (cabô de dados) para a troca de informações.

Do ponto de vista da aplicação, uma conexão consiste em dois fluxos independentes de direções opostas. **Uma conexão TCP é um fluxo de dados, e não de mensagens.** Isso significa que as fronteiras das mensagens não são preservadas de uma extremidade à outra. Quando uma aplicação passa dados para a entidade TCP, ela pode enviá-los imediatamente ou armazená-los em um **buffer\*** de acordo com suas necessidades.

**\*Buffer:** Área de memória temporária em que os dados são armazenados, aguardando o momento de serem transmitidos.

As entidades TCP transmissoras e receptoras trocam dados na forma de segmentos. Um segmento consiste em um cabeçalho fixo de 20 *bytes* mais uma parte opcional, seguido de zero ou mais *bytes* de dados. O *software TCP* decide qual deve ser o tamanho dos segmentos, podendo:



Acumular dados de várias escritas em um único segmento.



Dividir os dados de uma única escrita em vários segmentos.



## Atenção!

Cada segmento não pode ser superior à quantidade máxima de dados que um datagrama do protocolo IP é capaz de carregar.

O protocolo básico utilizado pelas entidades TCP é o de janela deslizante\*. Quando envia um segmento, o transmissor dispara um temporizador. Assim que ele chega ao destino, a entidade TCP receptora retorna um segmento (com ou sem dados segundo as circunstâncias) com um número de confirmação igual ao próximo número de sequência que ela espera receber. Se o temporizador do transmissor expirar antes de a confirmação ser recebida, o segmento será retransmitido.

**\*Janela deslizante:** Para aumentar a eficiência da transmissão, foram elaborados protocolos que permitem o envio de vários segmentos de dados mesmo sem a confirmação daqueles enviados anteriormente. O número máximo de dados que podem ser enviados sem que tenha chegado uma confirmação define o tamanho da janela de transmissão.

Conforme eles vão sendo confirmados, o ponto inicial da janela de transmissão desloca-se no sentido do fluxo de dados; por isso, ela é conhecida como “janela deslizante”. Protocolos que utilizam a janela de transmissão para o envio de dados são conhecidos como protocolos de janela deslizante.

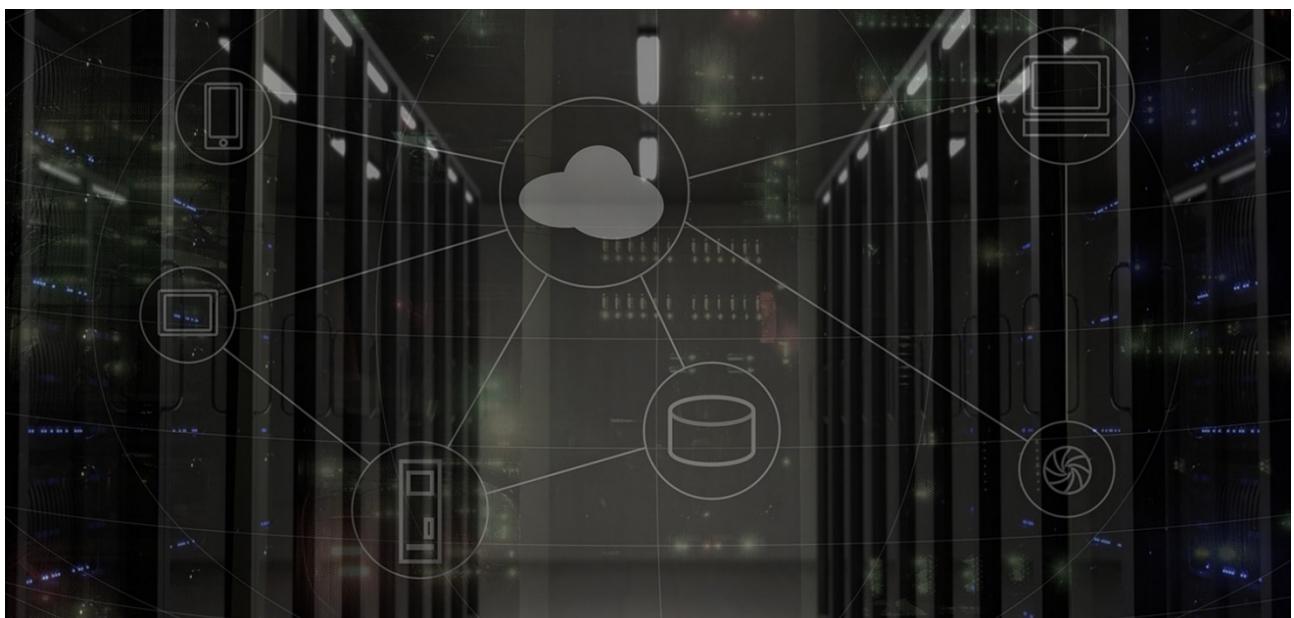


## Saiba mais

Para saber mais sobre janela deslizante, leia o capítulo 3.4.3 da obra de Kurose e Ross (2013).

Dessa forma, caso um segmento chegue ao destino e apresente erro em sua soma de verificação, o TCP simplesmente o descarta. Como não haverá confirmação de

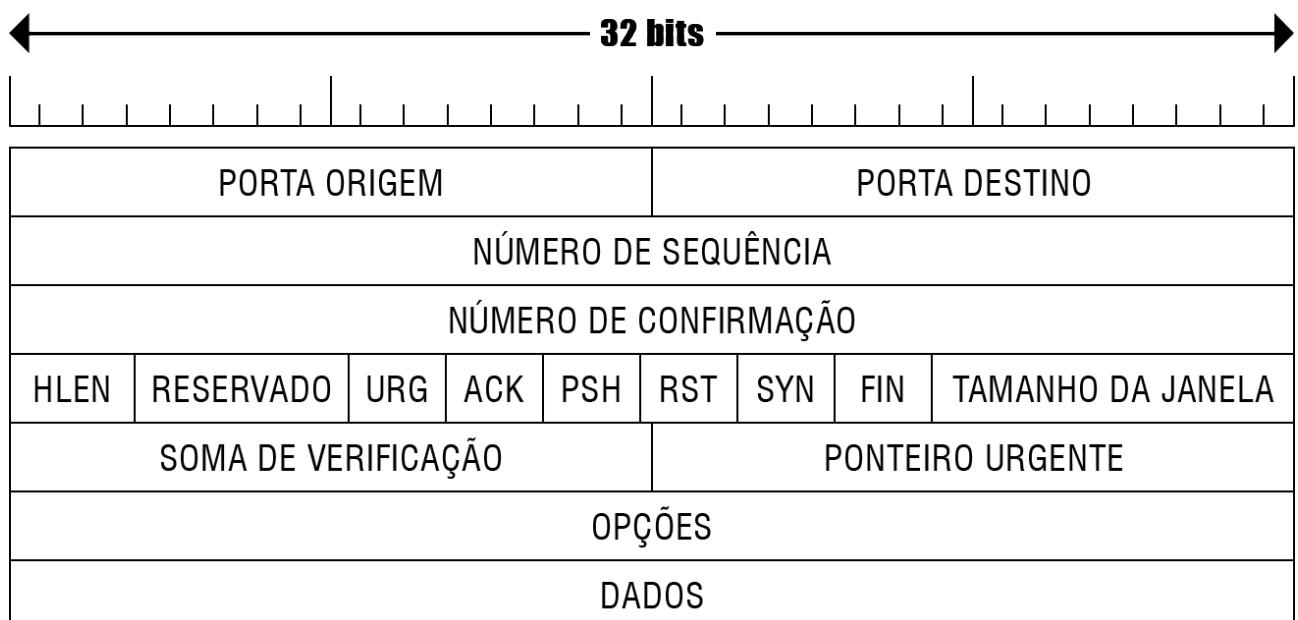
recebimento neste caso, a entidade TCP do transmissor entenderá que o segmento não chegou ao destino e providenciará sua **retransmissão**.



## CABEÇALHO DE SEGMENTO TCP

Cada segmento TCP começa com um cabeçalho de formato fixo – podendo ser seguido por opções de cabeçalho – de 20 *bytes*. Depois das opções, é possível haver 65.515 *bytes* de dados.

Válidos, os segmentos sem dados, em geral, são utilizados para confirmações e mensagens de controle. A figura a seguir mostra um exemplo de segmento TCP:



## Tabela 32 bits

**Porta origem e porta destino:** Identificam, respectivamente, os processos origem e destino nos hospedeiros. Da mesma forma que os campos porta origem e porta destino do UDP, eles permitem que aplicações compartilhem o uso do protocolo de transporte (multiplexação e demultiplexação).

**Número de sequência:** Indica o número de sequência do segmento, ou seja, em que posição (*byte*) dos dados ele deve ser colocado.

**Número de confirmação:** Especifica o próximo *byte* aguardado no fluxo contrário. Quando é enviado do processo A para o B, indica o próximo *byte* que A espera receber no fluxo de B para A. Quando informa que espera receber o *byte* N, fica implícito que todos os bytes até N-1 foram recebidos corretamente.

**HLEN:** Abreviação de *header length* (tamanho do cabeçalho), o HLEN informa o tamanho do cabeçalho em palavras de 32 *bits*.

**Reservado:** Campo não utilizado. Qualquer valor preenchido nele será desconsiderado.

**Campo de *flags* de 1 *bit* cada:** Um *flag* é um campo de 1 *bit* que pode possuir apenas os valores 0 e 1. Normalmente, 0 representa “desligado” e 1, “ligado”.

**Exemplo:** Quando o *flag* ACK do TCP está com o valor 1 (ligado), isso significa que o segmento TCP carrega um número de confirmação. Se ele estiver com 0 (desligado), é a indicação de que TCP não conta com tal número.

**URG:** O *urgent pointer* (ou ponteiro urgente) é usado para apontar que o segmento carrega dados urgentes.

**ACK:** O *acknowledgement* serve para indicar que o campo “número de confirmação” contém uma confirmação. Se estiver com valor 0, este campo deve ser desconsiderado.

**PSH:** O *push* se trata de uma solicitação ao receptor para entregar os dados à aplicação assim que eles chegarem em vez de armazená-los em um *buffer*.

**RST:** O *reset* é utilizado para reinicializar uma conexão que tenha ficado confusa devido a uma falha. Também serve para rejeitar um segmento inválido ou recusar uma tentativa de conexão.

**SYN:** Aplicado para estabelecer conexões.

**FIN:** Usado para encerrar uma conexão.

**Tamanho da janela:** O controle de fluxo no TCP é gerenciado por meio de uma janela deslizante de tamanho variável. O campo “tamanho da janela” indica quantos bytes podem ser enviados a partir do *byte* confirmado.

**Soma de verificação:** Utilizada para verificar se existem erros nos dados recebidos. Ela confere a validade tanto do cabeçalho quanto dos dados.

**Ponteiro urgente:** Válido somente se o *flag* URG estiver ativado, ele indica a porção de dados do campo de dados que contém os que são urgentes.

**Opções:** Projetadas como uma forma de oferecer recursos extras, ou seja, aqueles que não foram previstos pelo cabeçalho comum.

**Dados:** São os dados enviados pela camada superior. Neste campo, estão os que serão entregues na camada superior do hospedeiro destino.

## Gerenciamento de conexão TCP

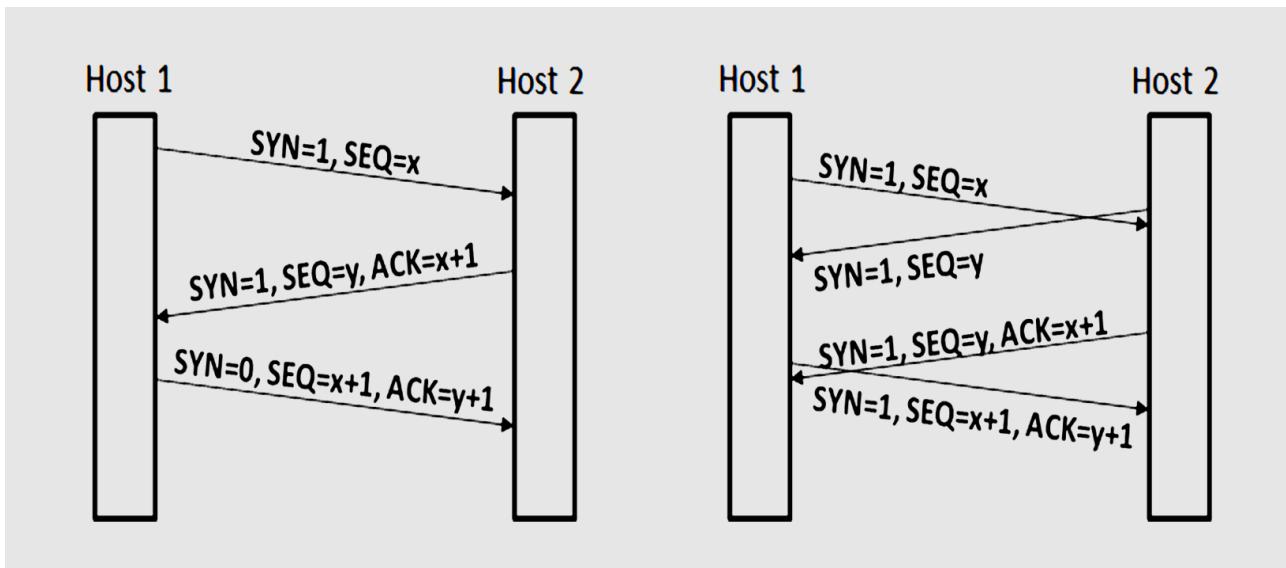
As conexões estabelecidas no TCP utilizam um esquema conhecido como *three way handshake*. Para estabelecer uma conexão, um lado aguarda passivamente, enquanto o outro solicita uma especificando o endereço de rede (endereço IP) e a porta com a qual deseja se conectar.



É enviado então um segmento TCP com o *bit* SYN ativado e o *bit* ACK desativado. Quando ele chega ao destino, a entidade TCP do destino verifica se existe um processo aguardando na porta destino. Se não existir, ela enviará uma resposta com o *bit* RST ativado para rejeitar a conexão.

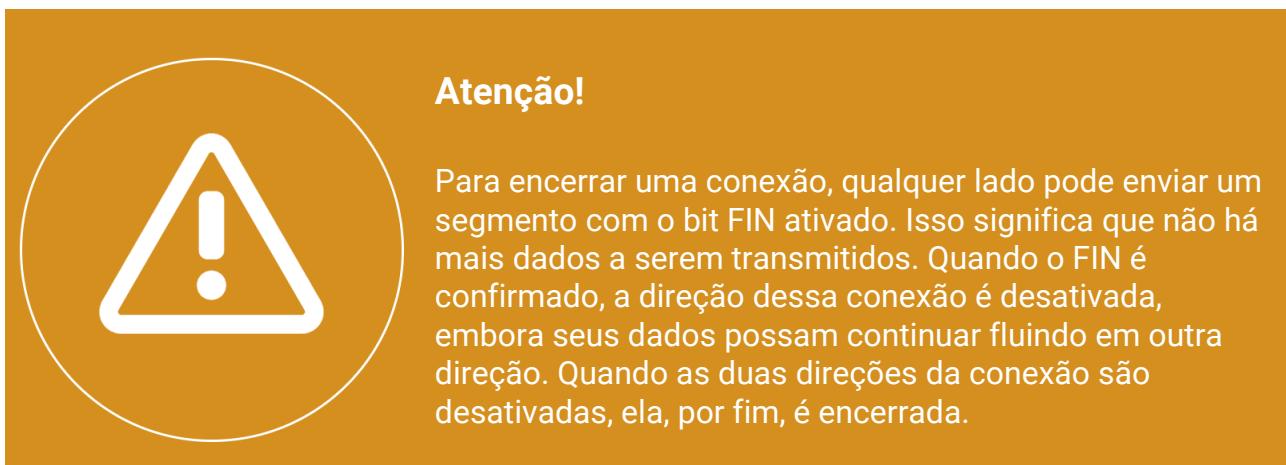
No entanto, se algum processo estiver na escuta dessa porta, a ele será entregue o segmento TCP recebido. Em seguida, tal processo poderá aceitar ou rejeitar a conexão. Se ele aceitar, um segmento de confirmação será retornado.

Esta figura ilustra a conexão entre dois hospedeiros:



Os segmentos SYN e SYN+ACK não podem transportar dados, porém consomem um número de segmento.

Apesar de conseguir fazer esse transporte, ACK só consumirá tal número quando ele surgir.





## Portas conhecidas

Para que uma aplicação possa acessar outra remota, é necessário conhecer o endereço do hospedeiro no qual ela se encontra. Ele serve, portanto, para que se consiga chegar ao hospedeiro remoto.

**Como o protocolo de transporte do destino consegue saber para qual de suas aplicações deve entregar a mensagem?** A resposta é o conceito de porta, que é responsável por identificar a aplicação no destino.

### Como podemos identificar a porta utilizada pela aplicação?

Existem duas saídas:

1

Empregar um sistema no qual a aplicação é registrada toda vez que inicializa para o cliente poder consultar sua porta.

2

Usar sempre o mesmo endereço de forma que as aplicações a iniciarem a conversação saibam de antemão com qual endereço trocar mensagens.

**Uma porta TCP ou UDP é identificada por um número inteiro de 16 bits.**

Para que um pacote chegue à aplicação de destino, é necessário que o transmissor saiba, de alguma forma, em que porta a aplicação está esperando a chegada do pacote. Para facilitar o trabalho dele, algumas aplicações esperam seus pacotes sempre em uma mesma porta: a “porta conhecida” da aplicação.



## Atenção!

A RFC 3232 define um repositório *on-line* no qual podem ser consultadas as portas conhecidas. No momento da criação deste documento, o repositório *on-line* estava definido como *service name and transport protocol port number registry*.

Esta tabela mostra as portas reservadas para algumas aplicações:

PORTE	APLICAÇÃO
7	echo
20	ftp-data
21	ftp
22	ssh
23	telnet
25	smtp
53	domain
69	tftp
80	http
110	pop-3
119	nntp
161	snmp
162	snmp-trap
443	https



## VERIFICANDO O APRENDIZADO

1. Sobre o protocolo UDP, marque a resposta correta.

- a) Verifica se a mensagem contém erros antes de entregá-la à aplicação destino.
- b) Garante a entrega da mensagem no destino.
- c) Não realiza a multiplexação e a demultiplexação.
- d) Não limita a quantidade máxima de dados que um segmento pode transportar.

### Comentário

**Parabéns!** A alternativa A está correta.

A camada de transporte deve oferecer confiabilidade na entrega das mensagens. Apesar de ser um protocolo que não garante a entrega dos dados, o UDP procura os erros para garantir, pelo menos, a entrega de uma mensagem sem a presença deles.

2. No protocolo TCP, o campo número de confirmação carrega:

- a) O número do último *byte* recebido com sucesso.
- b) O número do próximo *byte* esperado no sentido contrário.
- c) Uma solicitação para o receptor confirmar os dados recebidos.
- d) A quantidade de confirmações pendentes.

### Comentário

**Parabéns!** A alternativa **B** está correta.

O número de confirmação é a forma como o protocolo TCP faz para indicar quantos bytes foram recebidos com sucesso. Ele carrega o número no próximo byte esperado no sentido contrário, indicando, dessa maneira, que todos os bytes anteriores já foram recebidos corretamente.



## Considerações finais

# CONSIDERAÇÕES FINAIS

Vimos que, na camada de aplicação, são executadas as aplicações que os usuários desejam executar. Além disso, pontuamos que um desenvolvedor precisa se basear em um estilo de arquitetura para desenvolver seu *software* de aplicação.

Quando houve a apresentação da camada de transporte, também pudemos observar que existem dois serviços básicos: sem e com conexão. O desenvolvedor da aplicação precisa definir que tipo de serviço mais se adéqua a seu projeto.

Dessa forma, consideramos haver um forte relacionamento entre a camada de aplicação e a de transporte, pois o tipo de serviço de transporte escolhido pelo desenvolvedor gera um impacto direto no projeto da aplicação.

## CONTEUDISTA

Fábio Contarini Carneiro

## REFERÊNCIAS

ACM HYPERTEXT 91 CONFERENCE. **Proceedings**. San Antonio: Interaction Design Foundation, 1991.

COMER, D. E. **Redes de computadores e internet**. 6. ed. Porto Alegre: Bookman Editora, 2016.

COULOURIS, G. *et al.* **Sistemas distribuídos – conceitos e projeto**. 5. ed. Porto Alegre: Bookman Editora, 2013.

FOROUZAN, B. A. **Comunicação de dados e redes de computadores**. 4. ed. Porto Alegre: AMGH Editora, 2010.

INTERNET ASSIGNED NUMBERS AUTHORITY. Protocols registries. **Service name and transport protocol port number registry**. Califórnia: IANA, 2020.

KUROSE, J. F.; ROSS, K. W. **Redes de computadores e a internet**: uma abordagem top-down. 6. ed. São Paulo: Pearson, 2013.

SILBERSCHATZ, A.; GALVIN, P. B. **Sistemas operacionais – conceitos**. 5. ed. São Paulo: Prentice Hall, 2000.

SOARES, L. F. G. *et al.* **Redes de computadores** – das LANs, MANs e WANS às redes ATM. 2. ed. Rio de Janeiro: Campus, 1995.

TANENBAUM, A. S.; WETHERALL, D. **Redes de computadores**. 5. ed. São Paulo: Pearson, 2011.