Camadas de Sessão, Apresentação e Aplicação

Redes de Computadores

Profa. Kalinka Castelo Branco

Universidade de São Paulo

Junho de 2019

Roteiro

1. Modelo de Referência ISO/OSI
2. Aplicações
3. Arquitetura das Aplicações
4. Processos em Comunicação
5. Protocolos da Camada de Sessão
6. Protocolos da Camada de Apresentação

Camada de Sessão

Cuida dos processos que controlam a transferência dos dados, cuidando dos erros e administrando registros das transmissões.

- Serviços:
  - Controle de diálogo (quem deve transmitir em cada momento);
  - Gerenciamento de *tokens* (impede operações críticas de serem executadas ao mesmo tempo);
  - Sincronização (permite que uma transmissão continue do ponto onde parou).

Camadas Superiores do Modelo de Referência ISO/OSI

Camada de Apresentação

Serve como o tradutor de dados para a rede: está ligada à semântica e à sintaxe das informações transmitidas.

- Gerencia estruturas de dados abstratas:
  - Para que dois computadores possam se comunicar, as estruturas de dados a serem trocados podem ser definidas de maneira abstrata, com uma codificação padrão que será usada durante a conexão.
- É responsável pela compressão e criptografia dos dados.

Camadas Superiores do Modelo de Referência ISO/OSI

Camada de Aplicação

Engloba protocolos que realizam a comunicação fim-a-fim entre aplicações/processos de diferentes computadores.

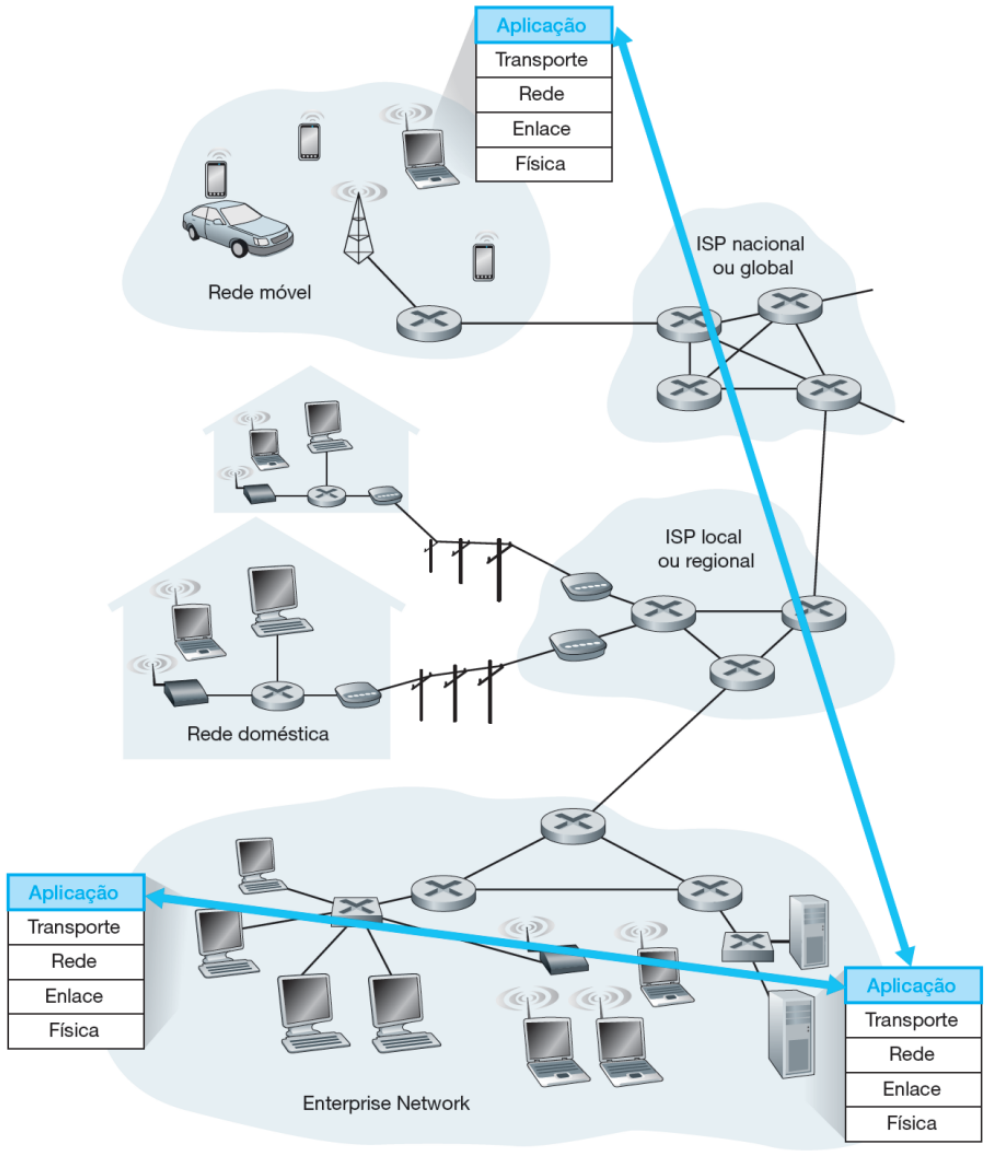
- Contém muitos protocolos comumente necessários pelos usuários;
- Frequentemente nos referimos às camadas de Sessão, Apresentação e Aplicação simplesmente como Camada de Aplicação:
  - As funções dessas três camadas são implementadas de modo bem integrado.

Algumas aplicações de rede

- E-mail;
- Web;
- Mensagens instantâneas;
- Login remoto;
- Compartilhamento de arquivos P2P;
- Jogos de rede multi-usuários;
- Vídeo-clipes;
- Voz sobre IP;
- Videoconferência em tempo real;
- Computação paralela em larga escala;
- Etc.

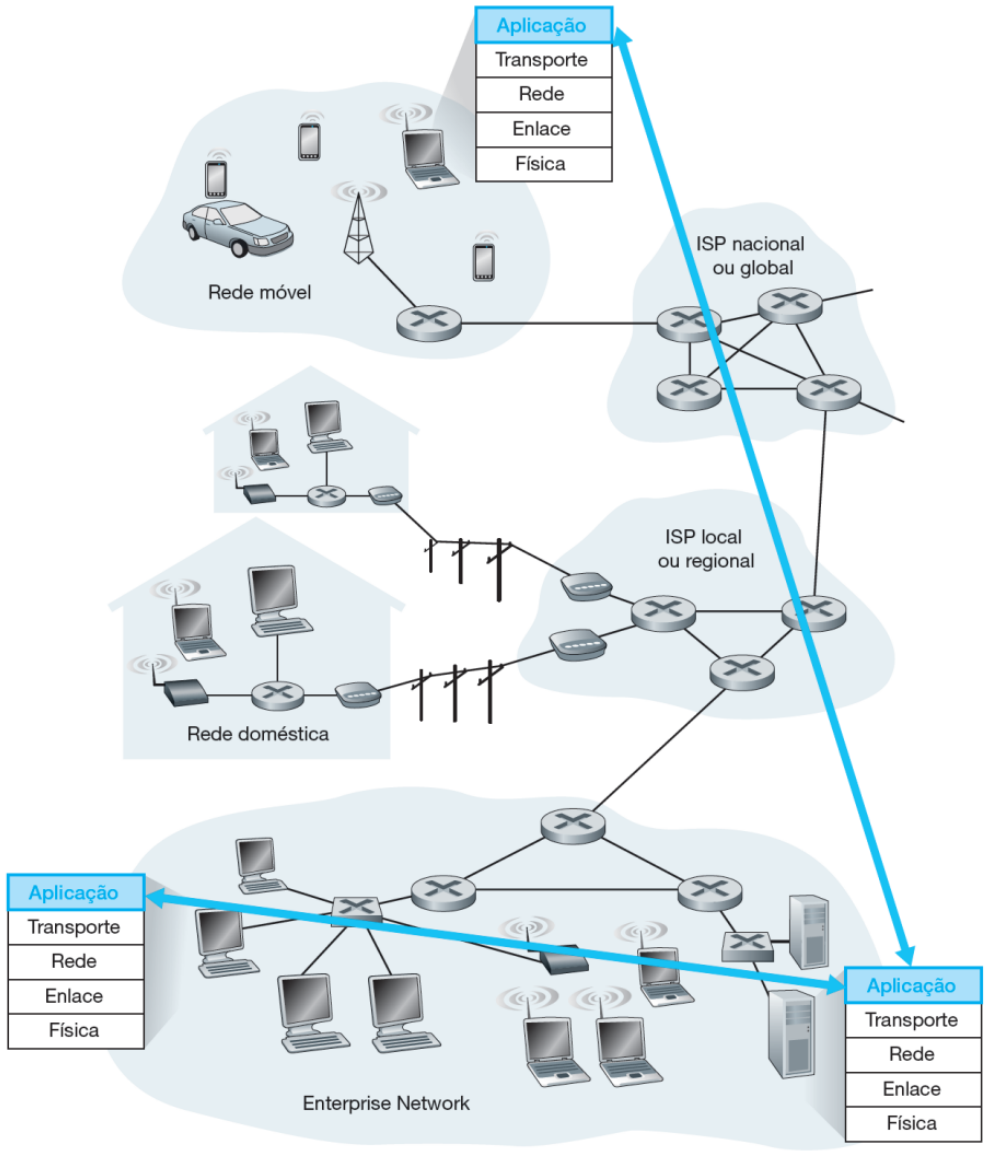
Criando uma aplicação de rede

- Programas que executam em diferentes sistemas finais e comunicam-se através da rede
  - Ex.: servidor Web se comunica com o navegador;



Criando uma aplicação de rede

- Programas não relacionados ao núcleo da rede:
  - Dispositivos do núcleo da rede não executam aplicações de usuários.
  - Aplicações nos sistemas finais permitem rápido desenvolvimento e disseminação.

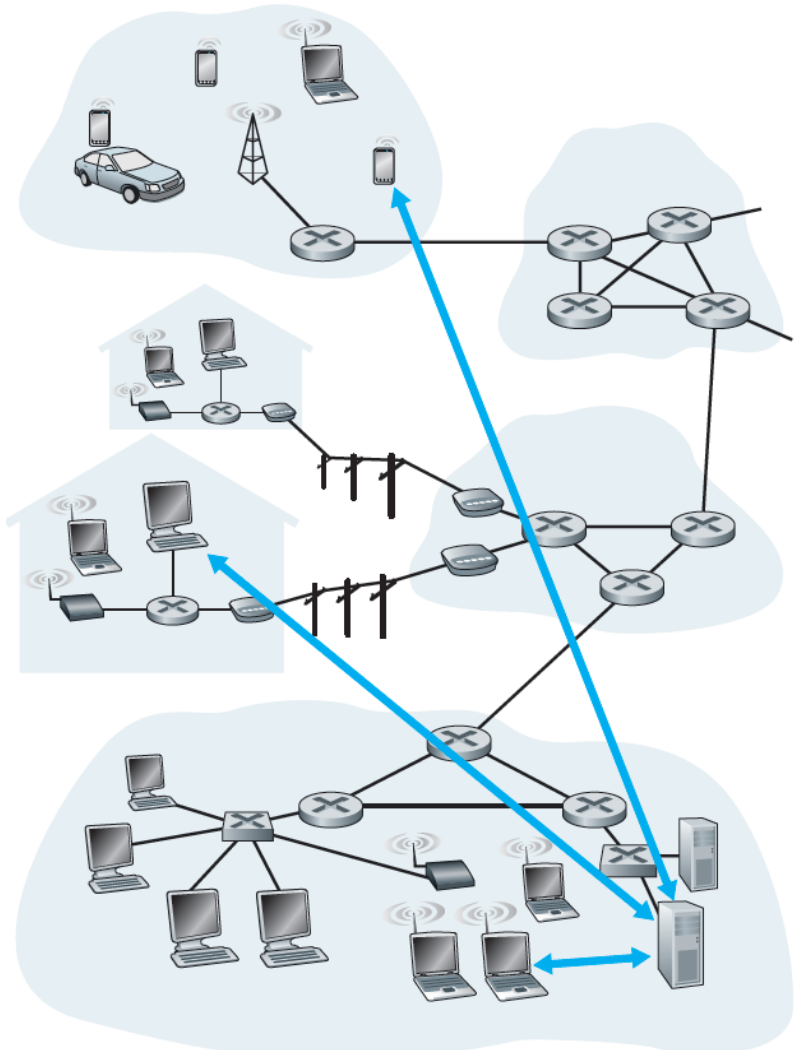


Arquiteturas das aplicações

- Cliente-servidor;
- *Peer-to-peer* (P2P);
- Híbrido de cliente-servidor e P2P.

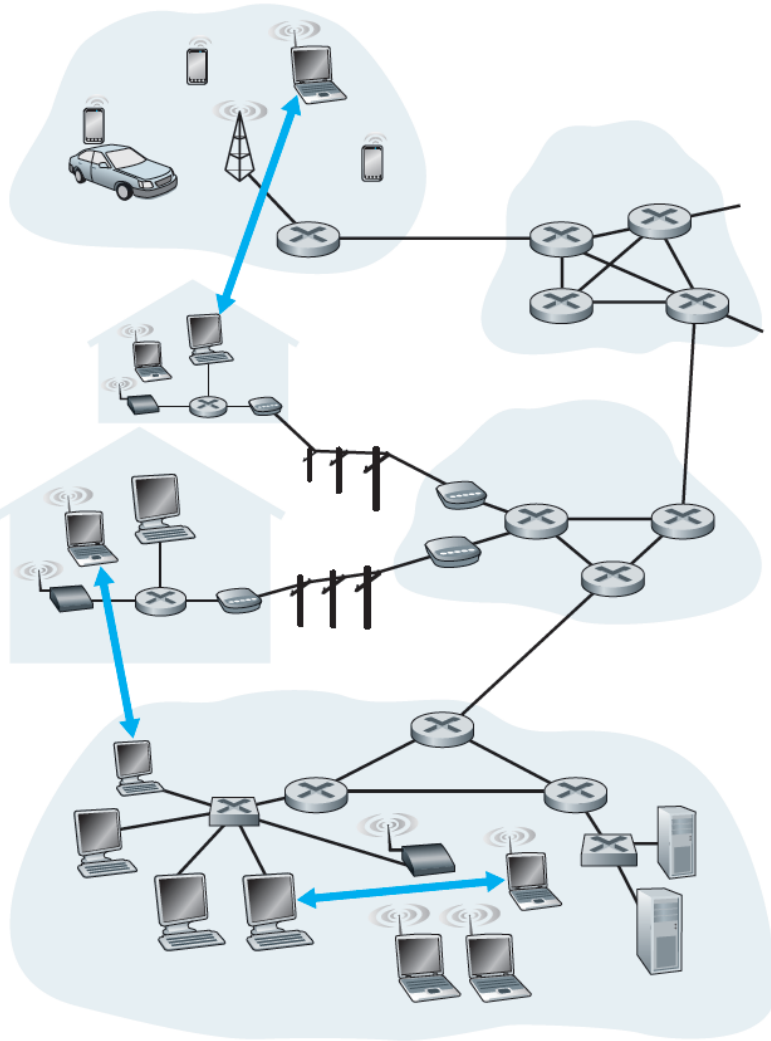
Arquitetura cliente-servidor

- **Servidor**:
  - Sempre ligado;
  - Endereço IP permanente;
  - Escalabilidade.
- **Cliente**:
  - Comunica-se com o servidor;
  - Pode estar conectado intermitentemente;
  - Pode ter endereços IP dinâmicos;
  - Não se comunica diretamente com outros clientes.



Arquitetura P2P pura

- Não há servidor sempre ligado;
- Sistemas finais arbitrários se comunicam diretamente;
- Pares estão conectados intermitentemente e mudam endereços IP;
- Exemplo: Gnutella;
- Altamente escalável;
- Porém, difícil de gerenciar.



Arquitetura híbrida de cliente-servidor e P2P

- Napster:
  - Transferência de arquivos P2P;
  - Busca de arquivos centralizada:
    - Pares registram conteúdo no servidor central;
    - Pares consultam o mesmo servidor central para localizar conteúdo.
- *Instant messaging*:
  - Conversa entre usuários P2P;
  - Localização e detecção de presença centralizadas:
    - Usuários registram o seu endereço IP junto ao servidor central quando ficam online;
    - Usuários consultam o servidor central para encontrar endereços IP dos contatos.

Processos em comunicação

- **Processo**: programa que executa num hospedeiro
  - Processos no mesmo hospedeiro se comunicam usando comunicação entre processos definida pelo sistema operacional (SO);
  - Processos em hospedeiros distintos se comunicam trocando mensagens através da rede.

Processos

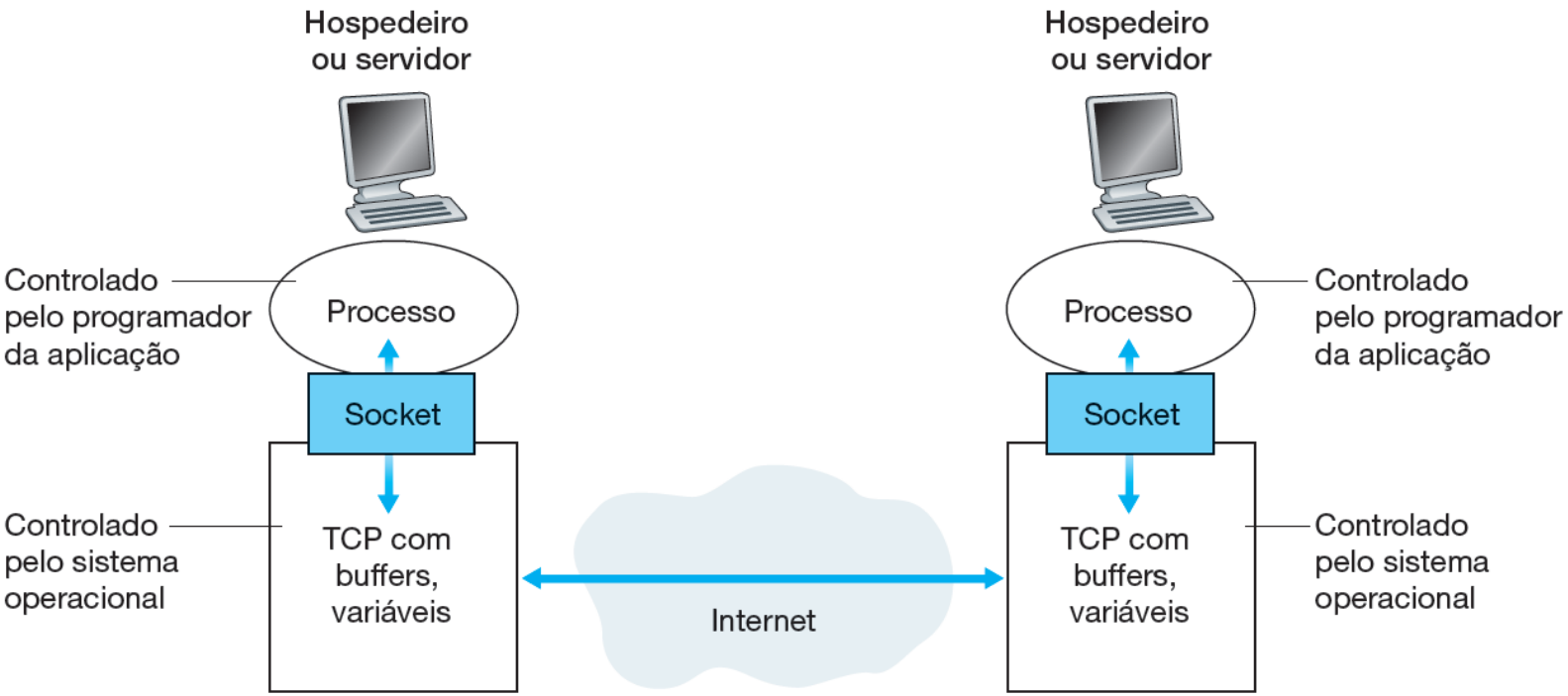
- Processo cliente: processo que inicia a comunicação;
- Processo servidor: processo que espera para ser contatado.

- Nota: aplicações com arquiteturas P2P possuem processos clientes e processos servidores.

Sockets

- Os processos enviam/recebem mensagens para/dos seus *sockets*;
- Um *socket* é análogo a uma porta:
  - Processo transmissor envia a mensagem através da porta;
  - O processo transmissor assume a existência da infraestrutura de transporte no outro lado da porta que faz com que a mensagem chegue ao *socket* do processo receptor.

Sockets


Hospedeiro ou servidor
Hospedeiro ou servidor
Controlado pelo programador da aplicação
Processo
Processo
Controlado pelo programador da aplicação
Socket
Socket
Controlado pelo sistema operacional
TCP com buffers, variáveis
TCP com buffers, variáveis
Controlado pelo sistema operacional
Internet

Endereçando os processos

- Para que um processo receba mensagens, ele deve possuir um identificador;
- Cada *host* possui um endereço IP único de 32 bits;
- **Pergunta**: o endereço IP do *host* no qual o processo está sendo executado é suficiente para identificar o processo?
- **Resposta**: não, muitos processos podem estar executando no mesmo *host*;
- O identificador inclui tanto o endereço IP quanto os números das portas associadas com o processo no *host*;
- Exemplo de números de portas:
  - Servidor HTTP: 80;
  - Servidor de E-mail: 25.

Protocolos da Camada de Sessão

- RPC (*Remote Procedure Call*) ou Chamada de Procedimento Remoto;
- RTP (*Real-time Transport Protocol*).

RPC

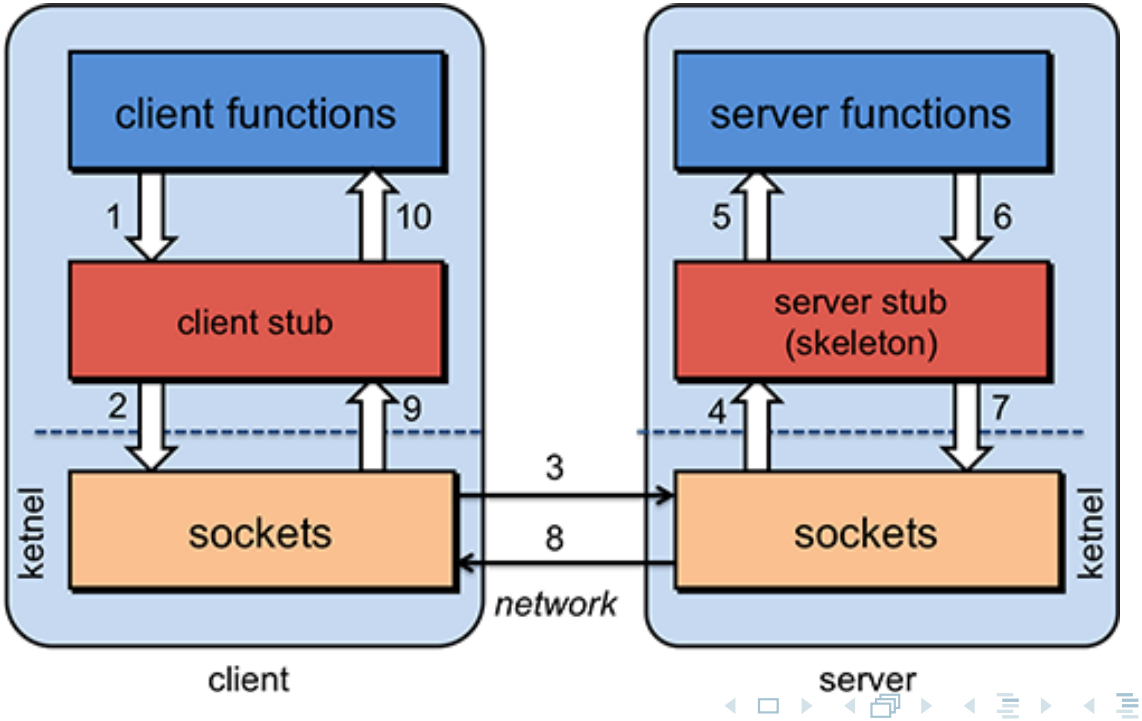
- O *Remote Procedure Call* (RPC) permite que clientes executem procedimentos em *hosts* remotos de forma que os detalhes da rede fiquem ocultos para o programador;
- Funcionamento:
  1. Processo no *host* 1 chama um procedimento no *host* 2 e fica suspenso;
  2. Processo no *host* 2 recebe a chamada de procedimento remoto e executa o procedimento;
  3. Quanto finalizada a execução, processo no *host* 2 envia a resposta para o *host* 1;
  4. *Host* 1 recebe a resposta como retorno do procedimento e retorna sua execução normalmente;

RPC

- Cliente é o processo que solicita a chamada de procedimento remoto:
  - Deve estar vinculado a um procedimento de biblioteca, chamado *stub* do cliente.
- Servidor é o processo que recebe a chamada e executa o procedimento remoto:
  - Deve estar vinculado a um procedimento de biblioteca, chamado *stub* do servidor ou *skeleton*.
- Os *stubs* são os procedimentos que utilizam as primitivas de *sockets* para as trocas de mensagens.

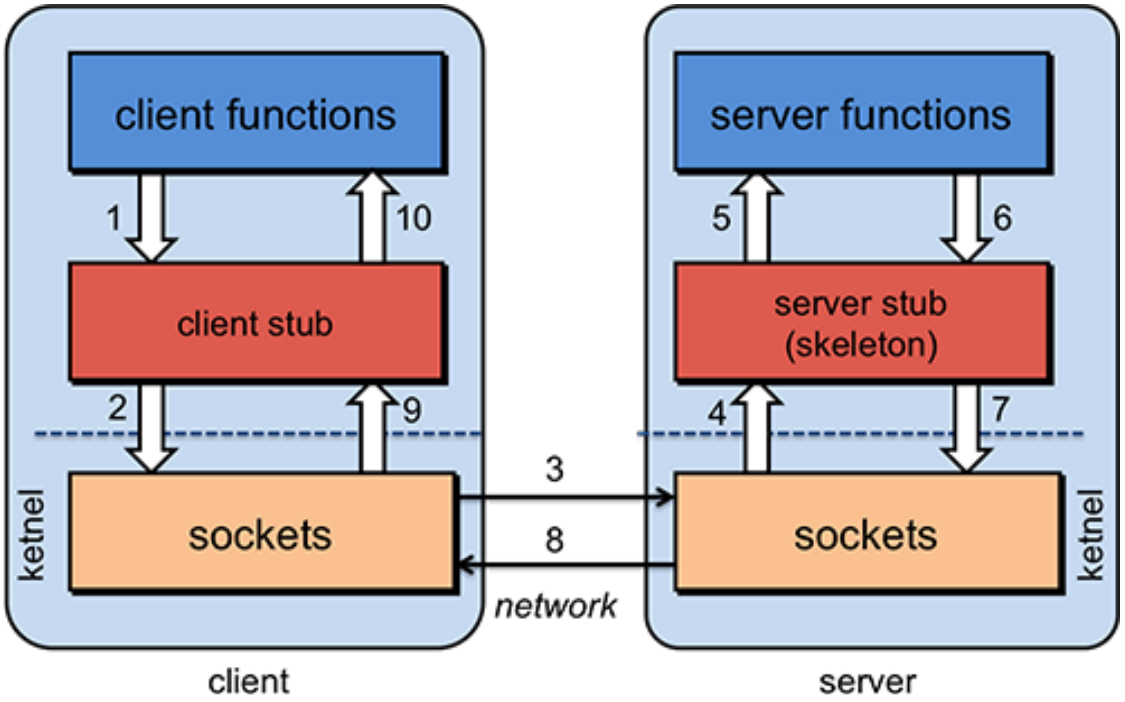
RPC

1. Cliente realiza a chamada de procedimento remoto (repassado para seu *stub*). Esse processo é visto como uma chamada de procedimento local para o programador. O *stub* cliente empacota os parâmetros, processo conhecido como empacotamento ou *marshaling*.



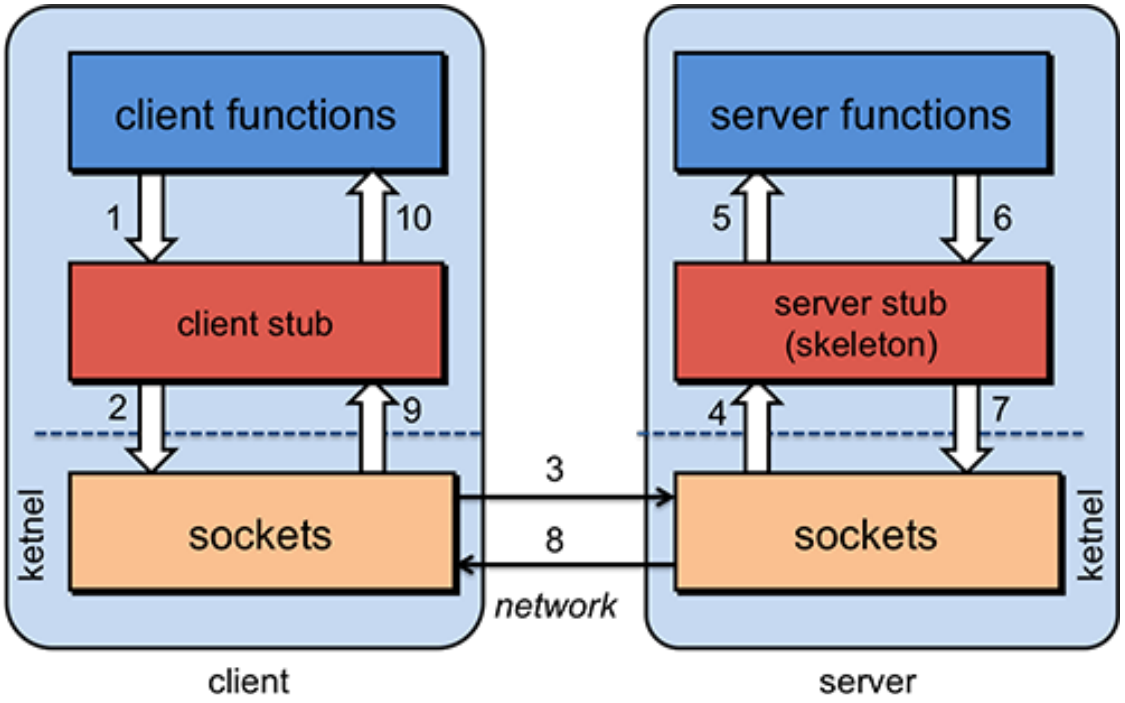
RPC

2. Envio da mensagem que contém a chamada do procedimento e os parâmetros como uma sequência de bytes (passaram também por serialização) por meio de *sockets*.



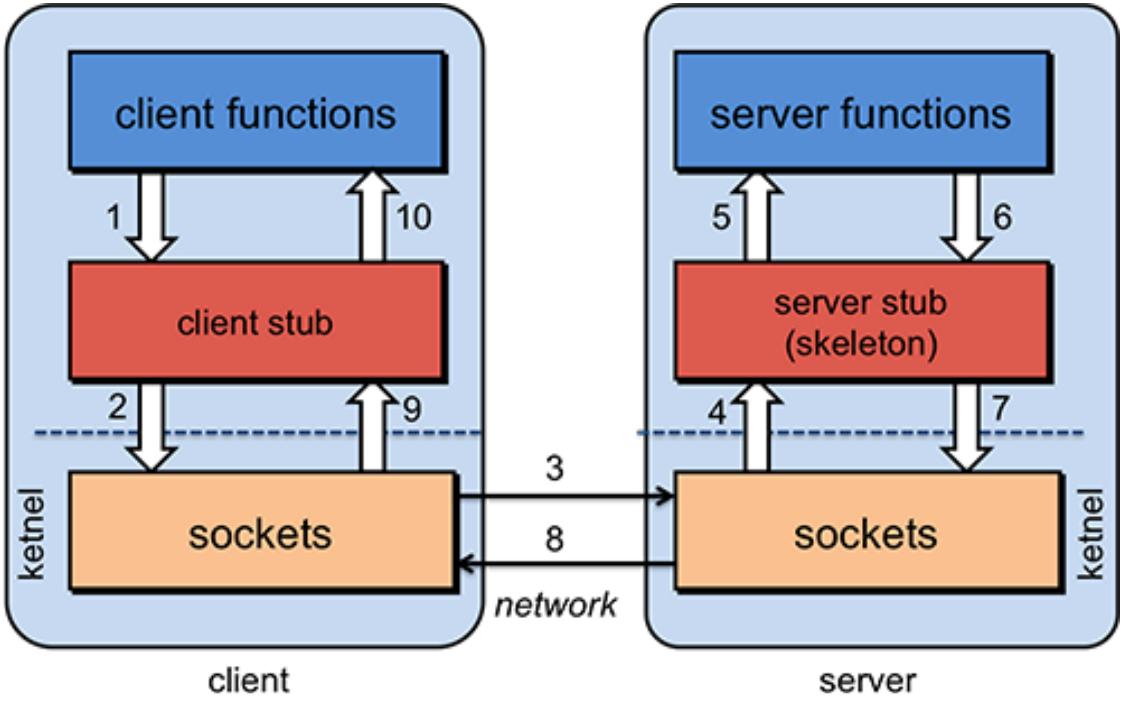
RPC

3. Envio da mensagem ao *host* remoto pelo protocolo de Camada de Transporte, que pode ser orientado à conexão ou não.



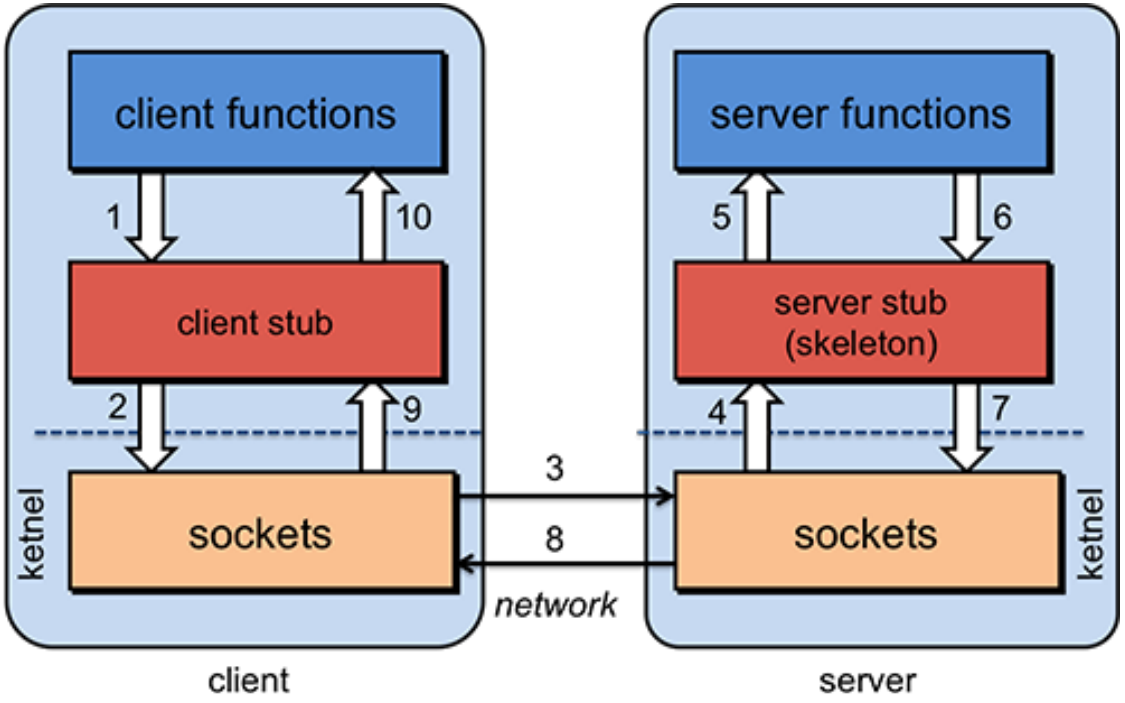
RPC

4. O *stub* do servidor (*skeleton*) recebe a mensagem por um *socket*, desempacota os parâmetros e converte para o formato específico de máquina.



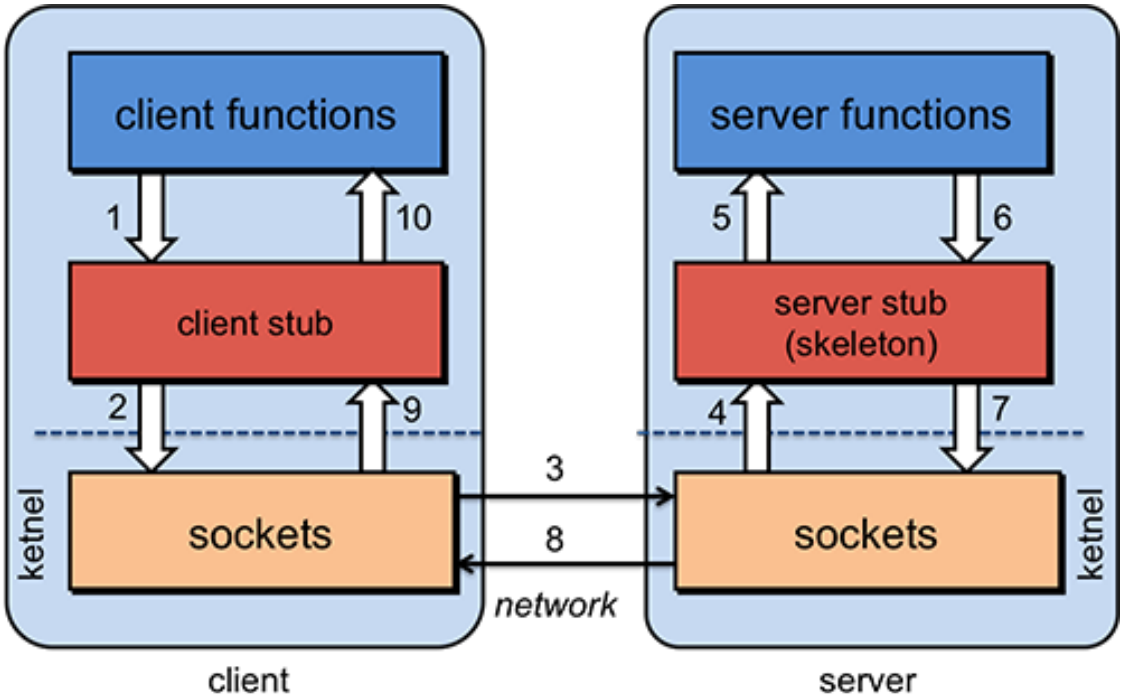
RPC

5. O *stub* do servidor chama a função do procedimento remoto disponível no servidor passando os argumentos definidos pelo cliente.



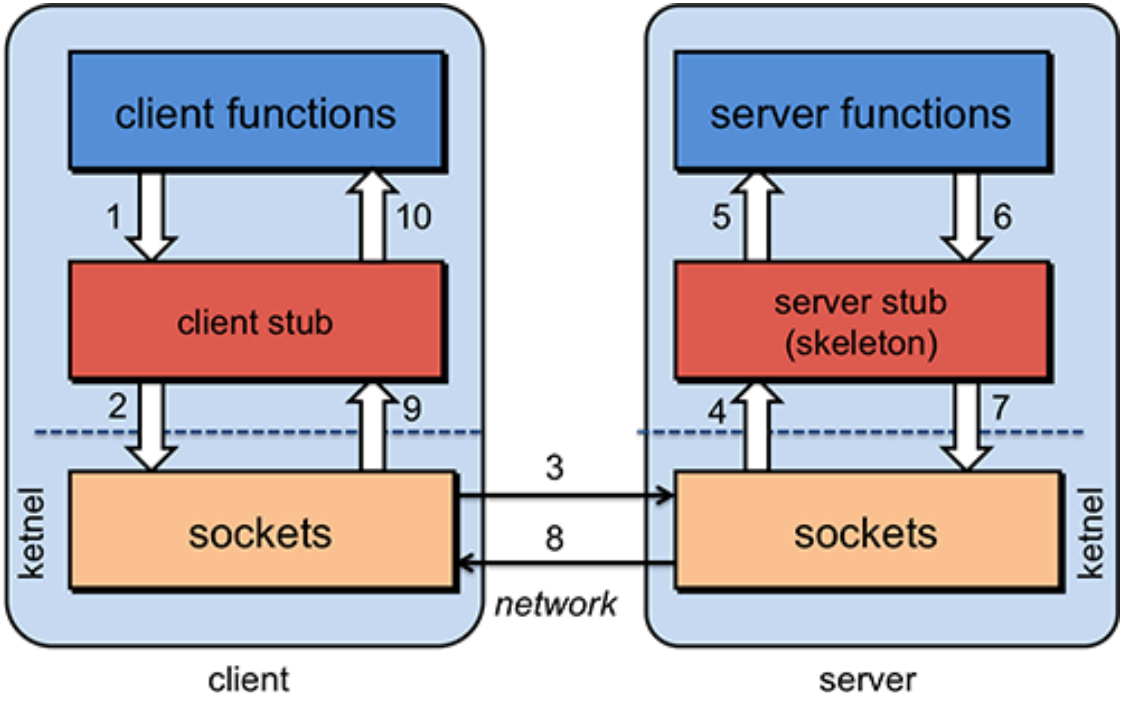
RPC

6. Quando finaliza a execução da função, os resultados retornam para o *stub* do servidor.



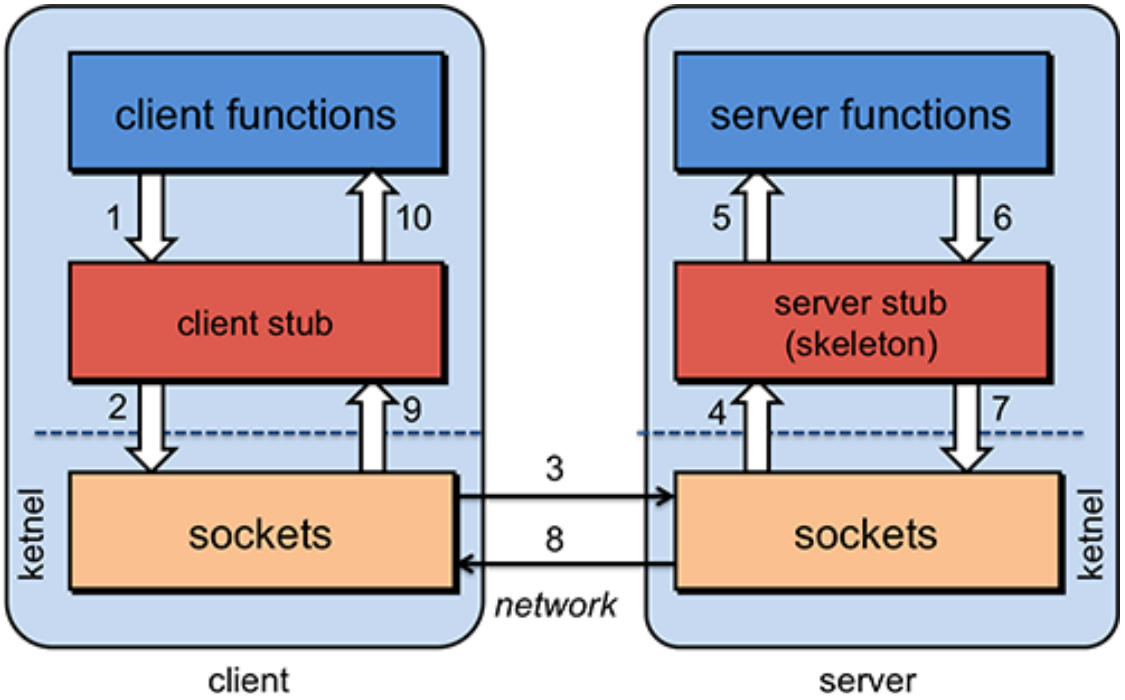
RPC

7. O *stub* do servidor serializa os resultados e realiza a chamada de primitiva do *socket*.



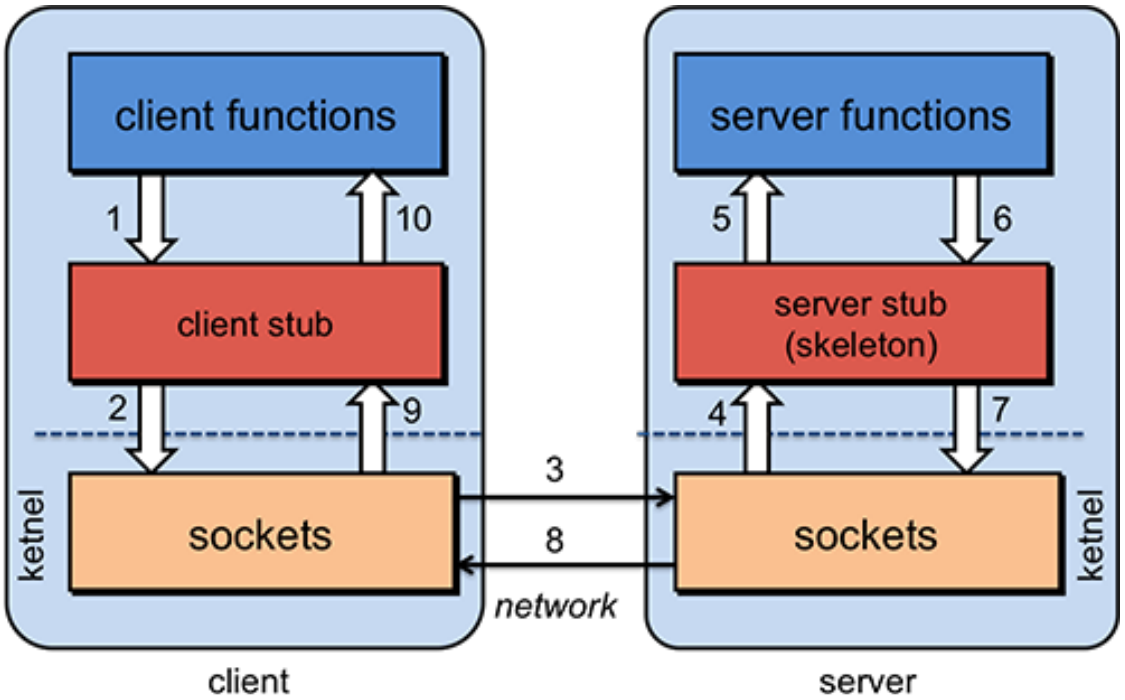
RPC

8. Os resultados são enviados como uma sequência de bytes pela rede para o cliente.



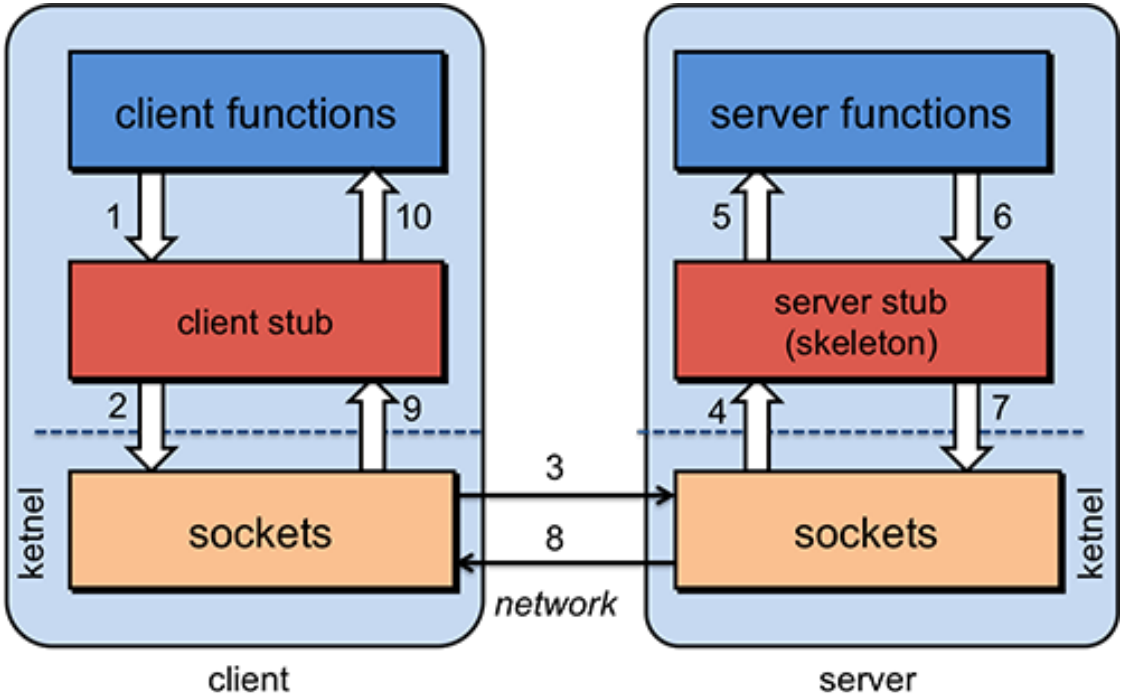
RPC

9. O *stub* do cliente recebe os resultados do seu *socket*.



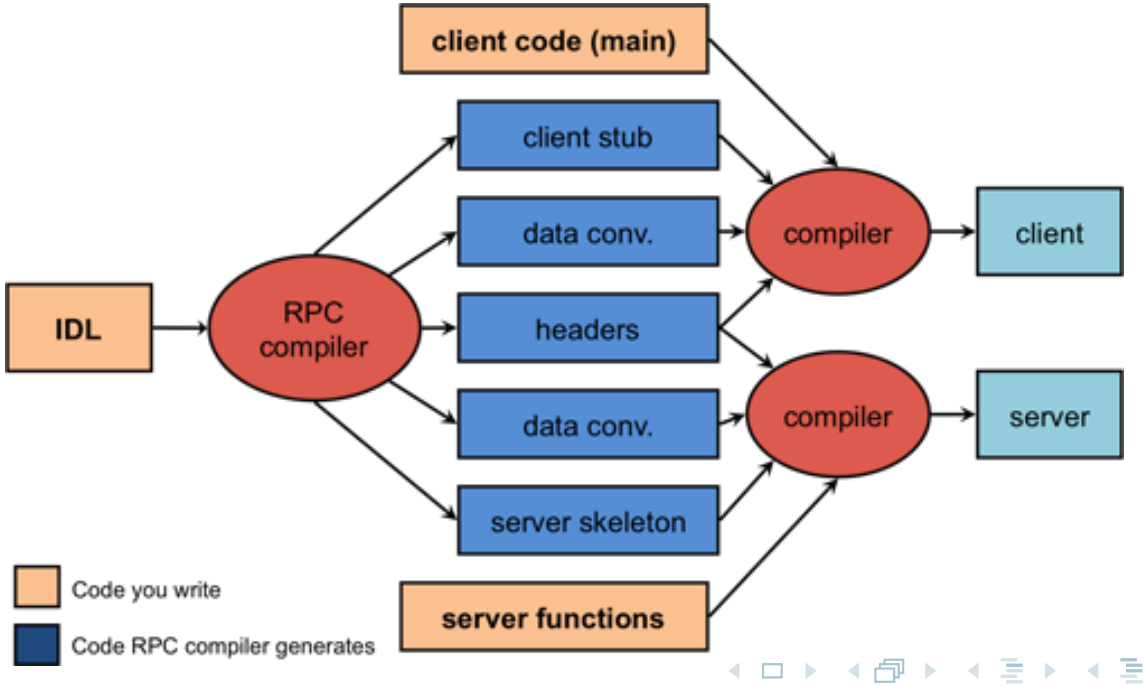
RPC

10. Resultados são remontados e retornados para a função do cliente.



RPC

- Linguagens mais populares não são adaptadas para chamadas de procedimento remoto, logo, é utilizado um compilador específico para gerar códigos:
  - O compilador utiliza uma definição da interface da chamada remota (*Interface Definition Language* ou IDL) como entrada e produz os *stubs* de cliente e servidor.



RPC

- Considerado um protocolo da Camada de Sessão e da Camada de Apresentação devido à reformatação dos parâmetros e dos resultados.
- Sistemas de RPC: ONC RPC ou Sun RPC, CORBA e Java RMI.
- Vantagens:
  - Facilidade para a programação de aplicações distribuídas por esconder detalhes de rede do programador;
  - Permite o uso da sintaxe tradicional de chamada de procedimentos.

RPC

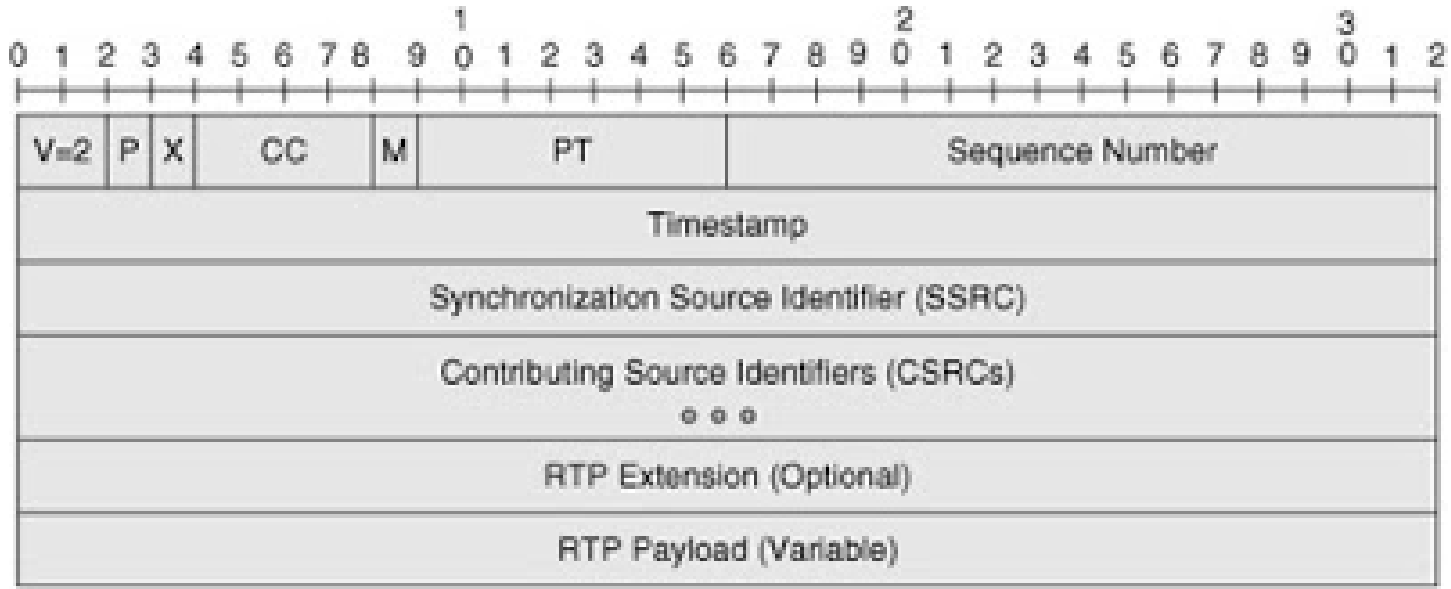
- Desvantagens:
  - Dificuldade para a passagem de parâmetros por referência, que resulta em restrições no formato dos procedimentos;
  - Não permite o uso de variáveis globais;
  - Necessário conhecer o servidor e as funções remotas previamente;
  - Desempenho inferior às chamadas locais, já que existe o *overhead* do empacotamento e do tempo de transmissão dos dados.
  - Problemas de segurança, já que os procedimentos podem acessar espaço de memória não permitido ou pode haver interceptação dos dados.

RTP

- O *Real-time Transport Protocol* (RTP) fornece serviço de transporte de *streamings* para aplicações multimídias:
  - Identificação do *payload*;
  - Numeração de sequência;
  - Timbre de hora (*timestamp*);
  - Monitoramento da rede e gerenciamento de QoS (*Quality of Service*).
- Utiliza o UDP para a transmissão de seus pacotes.

RTP

- PT (*Payload type*): identifica o formato do *payload*, como MP3;
- Número de sequência (*Sequence Number*): permite a ordenação dos pacotes e a detecção de pacotes perdidos;
- *Timestamp*: timbre de hora.
- SSRC: identificador da origem, permitindo multiplexar e demultiplexar vários fluxos.



RTP

- Uso de perfis para codificar e decodificar diferentes formatos de áudio e vídeo;
- O número de sequência permite a identificação de perda de pacotes e sua ordenação. Caso um pacote seja perdido, a aplicação pode executar uma aproximação por interpolação.
- Por meio do *timestamp* é possível detectar a diferença de tempo entre os pacotes, reproduzir cada amostra após um tempo correto e executar bufferizações.
- Outra vantagem do uso do *timestamp* é a sincronização de vários fluxos, como em uma transmissão de TV digital com um fluxo de vídeo e dois fluxos de áudio com dois idiomas diferentes.

RTP

- Geralmente associado ao RTP, é usado o *Real-Time Control Protocol* (RTCP) que monitora a rede, oferece serviços de sincronização e de interface para o usuário.
- Com informações a respeito da qualidade da rede, como latência (atraso) e *jitter* (variação do atraso), é possível que os algoritmos de codificação aumentem sua taxa de dados, resultando em melhor qualidade. E vice-versa, reduzindo sua taxa no caso de problemas na rede.
- Esse protocolo pode também executar tarefas de sincronização entre diferentes fontes de fluxo e nomear essas fontes para serem mostradas para o usuário.

Protocolos da Camada de Apresentação

- MIME (*Multipurpose Internet Mail Extensions*);
- SSL (*Secure Sockets Layer*);
- TLS (*Transport Layer Security*).

MIME

- O *Multipurpose Internet Mail Extensions* (MIME) é um padrão criado para permitir a inserção de componentes que não fossem caracteres ASCII em e-mails, por exemplo:
  - Outros caracteres além do ASCII (como de outros alfabetos);
  - Anexos binários, como arquivos de áudio, imagens e vídeos;
  - Textos enriquecidos com diferentes formatações de fontes, cores, entre outros;
- Em e-mails, é geralmente utilizado com o SMTP (protocolo de e-mails), mas também é usado com HTTP:
  - De acordo com o tipo do conteúdo, é possível especificar aplicativos de visualização corretos;

MIME

- Cabeçalho MIME:
  - **MIME-Version**: identifica a versão do MIME. Atualmente é usada a versão 1.0.
  - **Content-Type**: descreve o tipo e subtipos de dados. Exemplos: `text/plain`, `image/jpeg`, `audio/mp3`, `video/mp4`, e `application/msword`.
  - **Content-Transfer-Encoding:** define a codificação usada.
  - **Content-ID**: identificador único.
  - **Content-Description**: descrição do conteúdo.

MIME

Exemplo:

```
MIME-Version: 1.0
Content-Type: multipart/alternative;
boundary="--boundary text previous defined--"

--boundary text previous defined--
Content-Type: text/enriched;
charset="utf-8"
Content-Transfer-Encoding: quoted-printable

 This is the body text of a <bold>sample message</bold>.

--boundary text previous defined--
Content-Type: message/external-body;
access_type="anon-ftp";
site="bicycle.abcd.com";
directory="pub";
name="birthday.snd"
Content-Type: audio/basic
Content-Transfer-Encoding: base64

--boundary text previous defined--
```

SSL

- O *Secure Sockets Layer* (SSL) é um protocolo criptográfico para a comunicação segura com o protocolo TCP:
  - Muito usado com o protocolo HTTP, chamando-se de HTTPS;
  - Pode ser usado para qualquer aplicação sobre TCP.
- O SSL provê uma API com *sockets*, semelhante à API do TCP, disponibilizada por bibliotecas.

SSL

- Principais propriedades de uma comunicação segura:
  1. **Confidencialidade:** proteção contra descoberta ou interceptação não autorizada;
  2. **Integridade:** garantia de que não há modificação ou destruição da informação transmitida;
  3. **Disponibilidade:** assegura o acesso e uso rápido e confiável da informação;
  4. **Autenticação:** valida a identidade do usuário/entidade;
  5. **Não-repúdio:** impede que seja negada a autoria ou ocorrência de um envio ou recepção de uma informação.

SSL

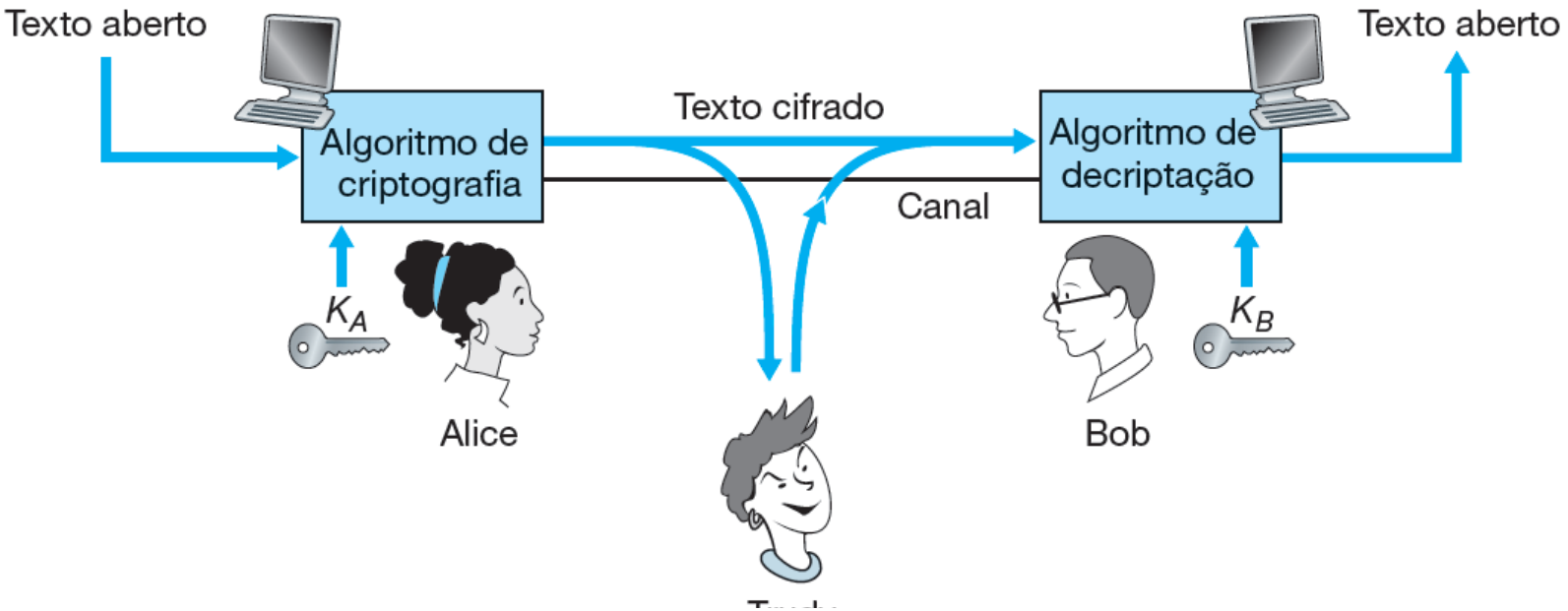
Criptografia

Conjunto de técnicas que permitem tornar incompreensível uma mensagem originalmente escrita com clareza, de forma a permitir que apenas o destinatário a decifre e a compreenda.

- SSL utiliza diversos algoritmos de criptografia para garantir confidencialidade e autenticação.

SSL

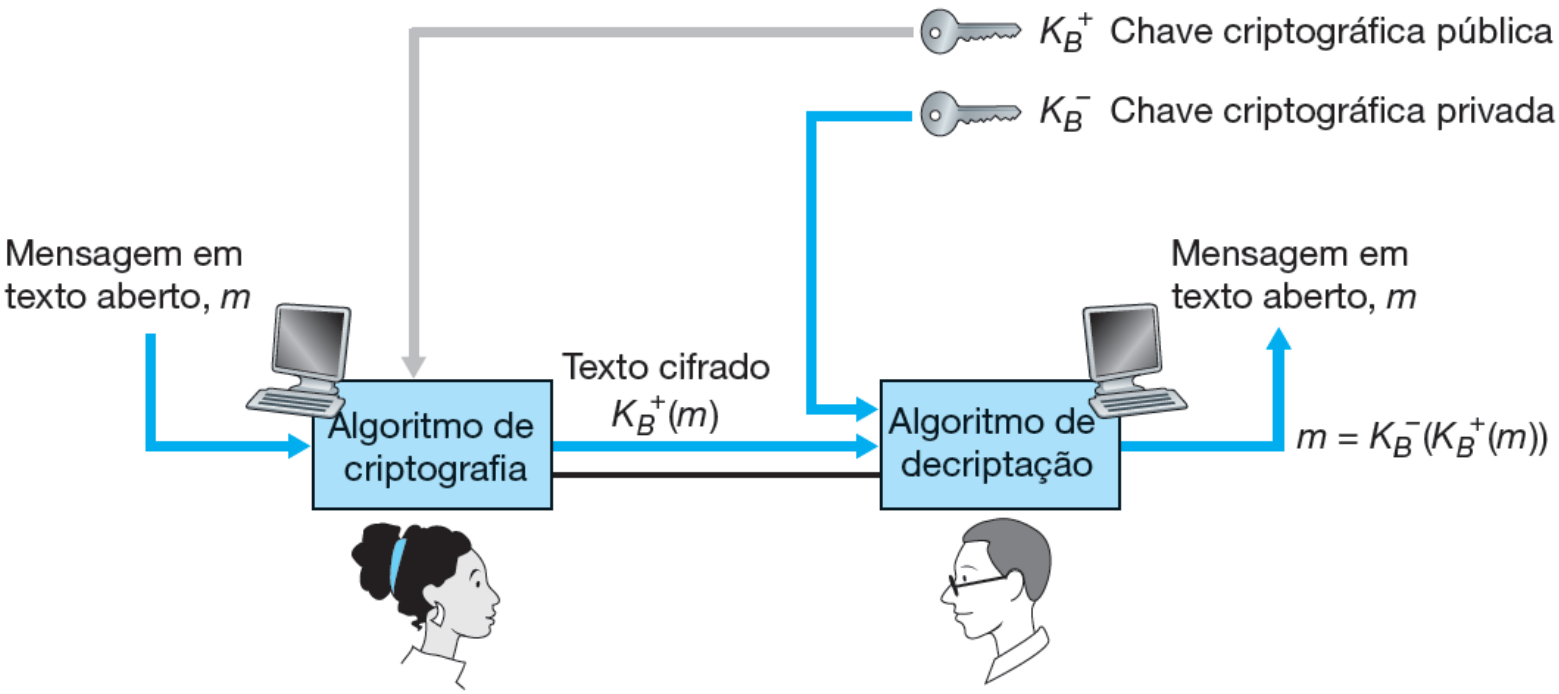
- Criptografia simétrica:



- Exemplos: AES, DES, 3DES.

SSL

- Criptografia assimétrica:



- Exemplos: RSA e ECC.

SSL

Função hash

Uma função que aceita uma mensagem de tamanho variável como entrada e produz um valor de hash de tamanho fixo.

Função de *hash* criptográfica

É um algoritmo para o qual é computacionalmente inviável descobrir um objeto de dados que seja mapeado para um resultado de hash pré-especificado ou dois objetos de dados que sejam mapeados para o mesmo resultado de hash.

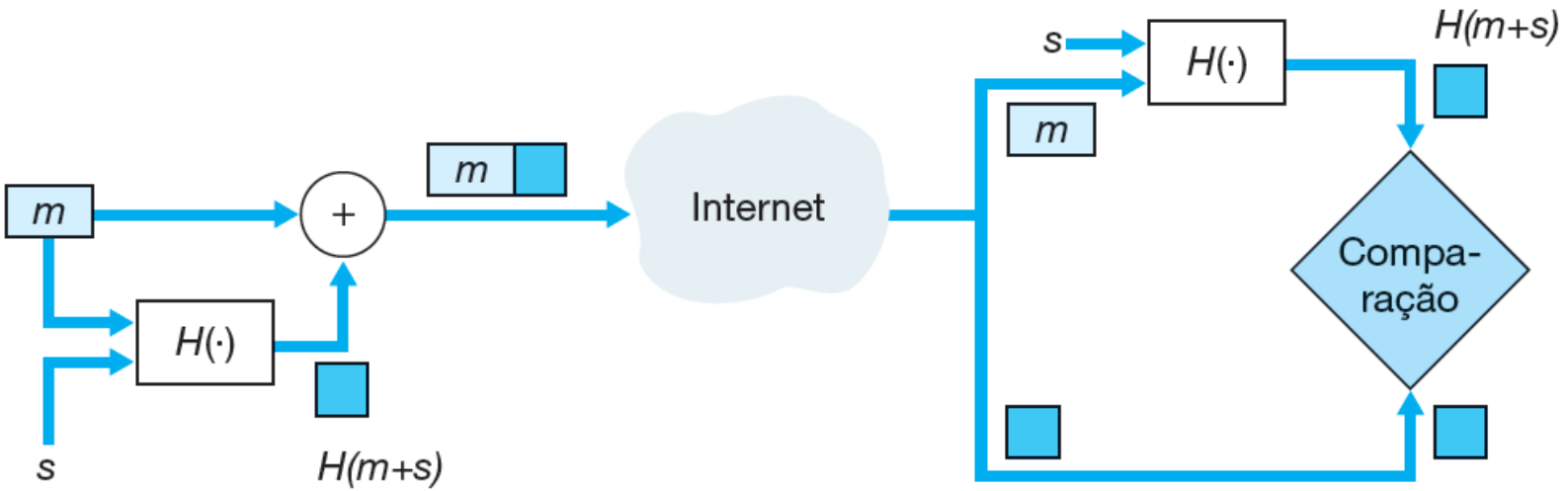
- SSL utiliza funções de hash criptográficas para garantir a integridade das mensagens.

SSL

- Possui três fases:
  - Apresentação (*handshake*): estabelecimento de algoritmos de criptografia e troca de chaves;
  - Derivação de chave: deriva-se chaves de criptografia e de Código de Autenticação de Mensagem (*Message Authentication Code* ou MAC);
  - Transferência de dados: troca de mensagens de forma segura.

SSL

- O MAC é o resultado da função hash criptográfica para a concatenação da mensagem com uma chave de autenticação e é sempre enviado com cada mensagem. No SSL, o MAC também contém o número de sequência dos pacotes.



Legenda:

m = Mensagem

s = Segredo compartilhado

SSL

- Pacotes SSL:
  - Tipo: indica se é uma mensagem de apresentação ou de dado;
  - Versão: versão do protocolo sendo usada;
  - Comprimento: comprimento é usado para extrair os dados;
  - Dados e MAC são criptografados com a chave de sessão E_B.

SSL

- Apresentação:
  1. Cliente envia uma lista de algoritmos de criptografia que ele suporta e um *nonce* (*Number Once*, um número que só é usado pelo protocolo uma única vez);
  2. Servidor escolhe um algoritmo simétrico, um algoritmo assimétrico e uma função hash criptográfica da lista do cliente e responde com suas escolhas, um certificado e um *nonce*;
  3. Cliente verifica o certificado, extrai a chave pública do servidor, gera um segredo pré-mestre e o envia criptografado com a chave pública do servidor;
  4. Acontece a fase de derivação de chave, onde são geradas as chaves de criptografia e de MAC a partir do segredo pré-mestre tanto pelo cliente quanto pelo servidor;
  5. Cliente envia o MAC de todas as mensagens de apresentação;
  6. Servidor envia o MAC de todas as mensagens de apresentação;

SSL

- Apresentação:
  1. Cliente envia uma lista de algoritmos de criptografia que ele suporta e um *nonce* (*Number Once*, um número que só é usado pelo protocolo uma única vez);
  2. Servidor escolhe um algoritmo simétrico, um algoritmo assimétrico e uma função hash criptográfica da lista do cliente e responde com suas escolhas, um certificado e um *nonce*;
  3. Cliente verifica o certificado, extrai a chave pública do servidor, gera um segredo pré-mestre e o envia criptografado com a chave pública do servidor;
  4. Acontece a fase de derivação de chave, onde são geradas as chaves de criptografia e de MAC a partir do segredo pré-mestre tanto pelo cliente quanto pelo servidor;
  5. Cliente envia o MAC de todas as mensagens de apresentação;
  6. Servidor envia o MAC de todas as mensagens de apresentação;

SSL

- Apresentação:
  1. Cliente envia uma lista de algoritmos de criptografia que ele suporta e um *nonce* (*Number Once*, um número que só é usado pelo protocolo uma única vez);
  2. Servidor escolhe um algoritmo simétrico, um algoritmo assimétrico e uma função hash criptográfica da lista do cliente e responde com suas escolhas, um certificado e um *nonce*;
  3. Cliente verifica o certificado, extrai a chave pública do servidor, gera um segredo pré-mestre e o envia criptografado com a chave pública do servidor;
  4. Acontece a fase de derivação de chave, onde são geradas as chaves de criptografia e de MAC a partir do segredo pré-mestre tanto pelo cliente quanto pelo servidor;
  5. Cliente envia o MAC de todas as mensagens de apresentação;
  6. Servidor envia o MAC de todas as mensagens de apresentação;

SSL

- Apresentação:
  1. Cliente envia uma lista de algoritmos de criptografia que ele suporta e um *nonce* (*Number Once*, um número que só é usado pelo protocolo uma única vez);
  2. Servidor escolhe um algoritmo simétrico, um algoritmo assimétrico e uma função hash criptográfica da lista do cliente e responde com suas escolhas, um certificado e um *nonce*;
  3. Cliente verifica o certificado, extrai a chave pública do servidor, gera um segredo pré-mestre e o envia criptografado com a chave pública do servidor;
  4. Acontece a fase de derivação de chave, onde são geradas as chaves de criptografia e de MAC a partir do segredo pré-mestre tanto pelo cliente quanto pelo servidor;
  5. Cliente envia o MAC de todas as mensagens de apresentação;
  6. Servidor envia o MAC de todas as mensagens de apresentação;

SSL

- Apresentação:
  1. Cliente envia uma lista de algoritmos de criptografia que ele suporta e um *nonce* (*Number Once*, um número que só é usado pelo protocolo uma única vez);
  2. Servidor escolhe um algoritmo simétrico, um algoritmo assimétrico e uma função hash criptográfica da lista do cliente e responde com suas escolhas, um certificado e um *nonce*;
  3. Cliente verifica o certificado, extrai a chave pública do servidor, gera um segredo pré-mestre e o envia criptografado com a chave pública do servidor;
  4. Acontece a fase de derivação de chave, onde são geradas as chaves de criptografia e de MAC a partir do segredo pré-mestre tanto pelo cliente quanto pelo servidor;
  5. Cliente envia o MAC de todas as mensagens de apresentação;
  6. Servidor envia o MAC de todas as mensagens de apresentação;

SSL

- Apresentação:
  1. Cliente envia uma lista de algoritmos de criptografia que ele suporta e um *nonce* (*Number Once*, um número que só é usado pelo protocolo uma única vez);
  2. Servidor escolhe um algoritmo simétrico, um algoritmo assimétrico e uma função hash criptográfica da lista do cliente e responde com suas escolhas, um certificado e um *nonce*;
  3. Cliente verifica o certificado, extrai a chave pública do servidor, gera um segredo pré-mestre e o envia criptografado com a chave pública do servidor;
  4. Acontece a fase de derivação de chave, onde são geradas as chaves de criptografia e de MAC a partir do segredo pré-mestre tanto pelo cliente quanto pelo servidor;
  5. Cliente envia o MAC de todas as mensagens de apresentação;
  6. Servidor envia o MAC de todas as mensagens de apresentação;

TLS

- O *Transport Layer Security* (TLS) é o sucessor do SSL e o protocolo de segurança recomendado atualmente, já que foram encontradas vulnerabilidades nas versões do SSL;
- O TLS utiliza algoritmos de criptografia e de MAC mais recentes e mais seguros, além de chaves maiores que também elevam o nível de segurança;
- A versão mais atual é o TLS 1.3 definido pelo RFC 8446, publicado em Agosto de 2018.