Resumo Ragnarok: Camada de Transporte!

Os serviços oferecidos pelo protocolo IP não oferecem confiabilidade. Problemas relacionados à congestionamento, perda e ordenação de pacotes não são tratados. Esse é um grande problema pois a camada de aplicação necessita prover um serviço confiável na entrega de dados para os usuários. Para tal, a camada de transporte tem diversos protocolos e funções para melhorar e corrigir erros das camadas abaixo da mesma - como a de redes.

Protocolos e serviços de transporte

* Fornecem comunicação lógica entre processos de aplicação em diferentes hospedeiros
* Os protocolos de transporte são executados nos sistemas finais
  + **Lado emissor:** quebra as mensagens da aplicação em segmentos e envia para a camada de rede
  + **Lado receptor:** remonta os 3 segmentos em mensagens e passa para a camada de aplicação
* Há mais de um protocolo de transporte disponível para as aplicações e nem todas implementam o mesmo conjunto de serviços:
  + Internet: TCP e UDP
  + TCP
    - Confiável: garante ordem de entrega
    - Controle de Congestionamento
    - Controle de Fluxo
    - Orientado à conexão
  + UDP
    - Não confiável: não garante ordem na entrega
    - Extensão do melhor esforço do IP
    - Apenas multiplexação

Identificação da aplicação no host

A camada de transporte oferece à camada de aplicação a função de endereçamento, onde os serviços são identificados pela sua porta (HTTP-80, FTP-20/21…) e uma conexão entre sua estação e outro host é feita através de um socket (IP+PORTA).

1. **Como cada máquina é identificada unicamente na Internet?**

*Número IP.*

1. **Como a entidade de rede (IP) identifica qual o protocolo de transporte está sendo utilizado?**

*Tipo de protocolo está indicado no cabeçalho IP.*

1. **Dentro do host, como a entidade de transporte identifica qual aplicação está sendo utilizada?**

*Cada aplicação tem uma “Porta” única no host. Porta é identificada no pacote IP.*

1. **Como uma aplicação cliente sabe qual a porta de uma aplicação servidora para poder enviar pacotes?**

*Alguns serviços têm números de portas já convencionadas (portas “bem conhecidas”).*

**Números de portas**

**1-255** Reservadas para serviços padrão portas “bem conhecidas”.

**256-1023** Reservado para serviços Unix.

**1-1023** Somente podem ser usadas por super-usuário.

**1024-4999** Usadas por processos de sistema e de usuário.

**5000-** Usadas somente por processos de usuário.

Demultiplexação/Multiplexação

* **Demultiplexação no hospedeiro receptor:**
  + Entrega os segmentos recebidos ao socket correto
* **Multiplexação no hospedeiro receptor:**
  + Coleta dados de múltiplos sockets, envelopa os dado com cabeçalho (usado depois para demultiplexação).
* **Computador recebe datagramas IP** 
  + Cada datagrama possui endereço IP de origem e IP de destino
  + Cada datagrama carrega 1 segmento da camada de transporte
  + Cada segmento possui números de porta de origem e destino (lembre-se: números de porta bem conhecidos para aplicações específicas)
  + O hospedeiro usa endereços IP e números de porta para direcionar o segmento ao socket apropriado.
* **UDP**:
* Socket **UDP** identificado por dois valores: (endereço IP de destino, número da porta de destino)
  + Quando o hospedeiro recebe o segmento **UDP**:
    - Verifica o número da porta de destino no segmento
    - Direciona o segmento **UDP** para o socket com este número de porta
  + Datagramas com IP de origem diferentes e/ou portas de origem diferentes são direcionados para o mesmo socket.
* **TCP**:
* Socket **TCP** identificado por 4 valores:
  + Endereço IP de origem
  + Endereço porta de origem
  + Endereço IP de destino
  + Endereço porta de destino
* Hospedeiro receptor usa os quatro valores para direcionar o segmento ao socket apropriado
* Hospedeiro servidor pode suportar vários sockets **TCP** simultâneos:
  + Cada socket é identificado pelos seus próprios 4 valores
  + Servidores possuem sockets diferentes para cada cliente conectado.

UDP: User Datagram Protocol

O UDP é um protocolo da camada de transporte não confiável e não-orientado à conexão. Fornece apenas os serviços de endereçamento e fragmentação, não provendo confiabilidade (controle de fluxo, erro, congestionamento). Isso indica que o UDP não adiciona serviços ao protocolo IP. O que nos leva então a utilizar o UDP?

O UDP por ser mais simples possui um cabeçalho menor gerando um menor overhead. Ideal para algumas aplicações onde a velocidade é mais útil que a confiabilidade como aplicações multimídia. Afinal não faz sentido algum receber um trecho de um arquivo música que já passou.

Além de aplicações multimídia, o UDP é utilizado também pelo TFTP (Trivial File Transfer Protocol), RIP (Routing Information Protocol), SNMP (Simple Network Management Protocol) e DNS (Domain Name System).

* Serviço “best effort”, segmentos **UDP** podem ser:
  + **Perdidos** ou **Entregues** fora de ordem para a aplicação
* Sem conexão:
  + Não há apresentação entre o **UDP** transmissor e o receptor.
  + Cada segmento **UDP** é tratado de forma independente dos outros.
* Por que existe um **UDP**?
  + Não há estabelecimento estabelecimento de conexão conexão (que possa resultar resultar em atrasos atrasos).
  + Simples: não há estado de conexão nem no transmissor, nem no receptor.
  + Cabeçalho de segmento reduzido.
  + Não há controle de congestionamento: **UDP** pode enviar segmentos tão rápido quanto desejado (e possível).
* Muito usado por aplicações de multimídia contínua (**streaming**) **UDP**:
  + Tolerantes à perda.
  + Sensíveis à taxa.
* Transferência confiável sobre **UDP**: acrescentar confiabilidade na camada de aplicação
  + Recuperação de erro específica de cada aplicação.

**UDP Checksum**

Objetivo: detectar “erros” (ex.: bits trocados) no segmento transmitido

* **Transmissor**:
  + Trata o conteúdo do segmento como seqüência de inteiros de 16 bits
  + Checksum: soma (complemento de 1 da soma) do conteúdo do segmento
  + Transmissor coloca o valor do checksum no campo de checksum do UDP
* **Receptor**:
  + Computa o checksum do segmento recebido
  + Verifica se o checksum calculado é igual ao valor do campo checksum:
  + NÃO - erro detectado
  + SIM - não há erros. Mas talvez haja erros apesar disso? Mas depois…

Princípios de transferência confiável de dados

Importante nas camadas de aplicação, transporte e enlace. Características dos canais não confiáveis determinarão a complexidade dos protocolos confiáveis de transferência de dados (rdt).

**Transferência confiável usando um canal com erro de bits**

* Canal subjacente pode trocar valores dos bits num pacote
  + Checksum para detectar erros de bits
* A questão: como recuperar esses erros:
  + **Reconhecimentos (ACKs):** receptor avisa explicitamente ao transmissor que o pacote foi recebido corretamente
  + **Reconhecimentos negativos (NAKs):** receptor avisa explicitamente ao transmissor que o pacote tem erros
  + Transmissor reenvia o pacote quando da recepção de um NAK
* Mecanismos necessários:
  + Detecção de erros
  + Retorno do receptor: mensagens de controle
  + (ACK, NAK) rcvr->sender

**Transferência confiável usando um canal com erro de bits e perdas**

**O que acontece se o ACK/NAK é corrompido ou perdido?**

* Transmissor não sabe o que aconteceu no receptor!
* Transmissor deve esperar durante um tempo razoável pelo ACK e se não recebê-lo deve retransmitir a informação
  + Não pode apenas retransmitir: possível **duplicata**

**Tratando duplicatas:**

* Transmissor acrescenta número de seqüência em cada pacote
* Transmissor reenvia o último pacote se ACK/NAK for perdido
* Receptor descarta (não passa para a aplicação) pacotes duplicados.

**Estratégias de Retransmissão**

* Conhecidos como algoritmos ou protocolos Automatic Repeat Request (**ARQ**)
* Questões de projeto:
  + Como o receptor requisita uma retransmissão?
  + Como a fonte sabe quando retransmitir?
* Desempenho e exatidão
* Para simplificar as explicações assumimos comunicações do tipo ponto-a-ponto
* Mesmas soluções adotadas pela camada de enlace:
  + Verificação de erros (checksum)
  + Retransmissão
  + Temporização
  + Número de sequência
  + Pipelining
  + Janela deslizante

O Protocolo TCP

O TCP é o protocolo da camada de transporte da arquitetura Internet responsável em oferecer confiabilidade na transmissão. O TCP fornece um **serviço orientado à conexão, confiável e full-duplex** para os serviços de aplicação.

O TCP é **orientado a conexão** – Para ter o controle dos pacotes enviados e conseguir efetuar a fragmentação, o TCP precisa que os usuários finais tenham o controle do que está sendo enviado. O protocolo TCP especifica três fases durante uma conexão: estabelecimento da ligação, transferência e término de ligação. Para estabelecimento da conexão o TCP necessita que:

*“O cliente inicia a ligação enviando um pacote TCP com a flag SYN activa e espera-se que o servidor aceite a ligação enviando um pacote SYN+ACK. Se, durante um determinado espaço de tempo, esse pacote não for recebido ocorre um timeout e o pacote SYN é reenviado. O estabelecimento da ligação é concluído por parte do cliente, confirmando a aceitação do servidor respondendo-lhe com um pacote ACK”.*

Efetua desconexão “suave” (**Graceful Connection Shutdown**) – O TCP só encerra a conexão depois de entregar os dados ao receptor.

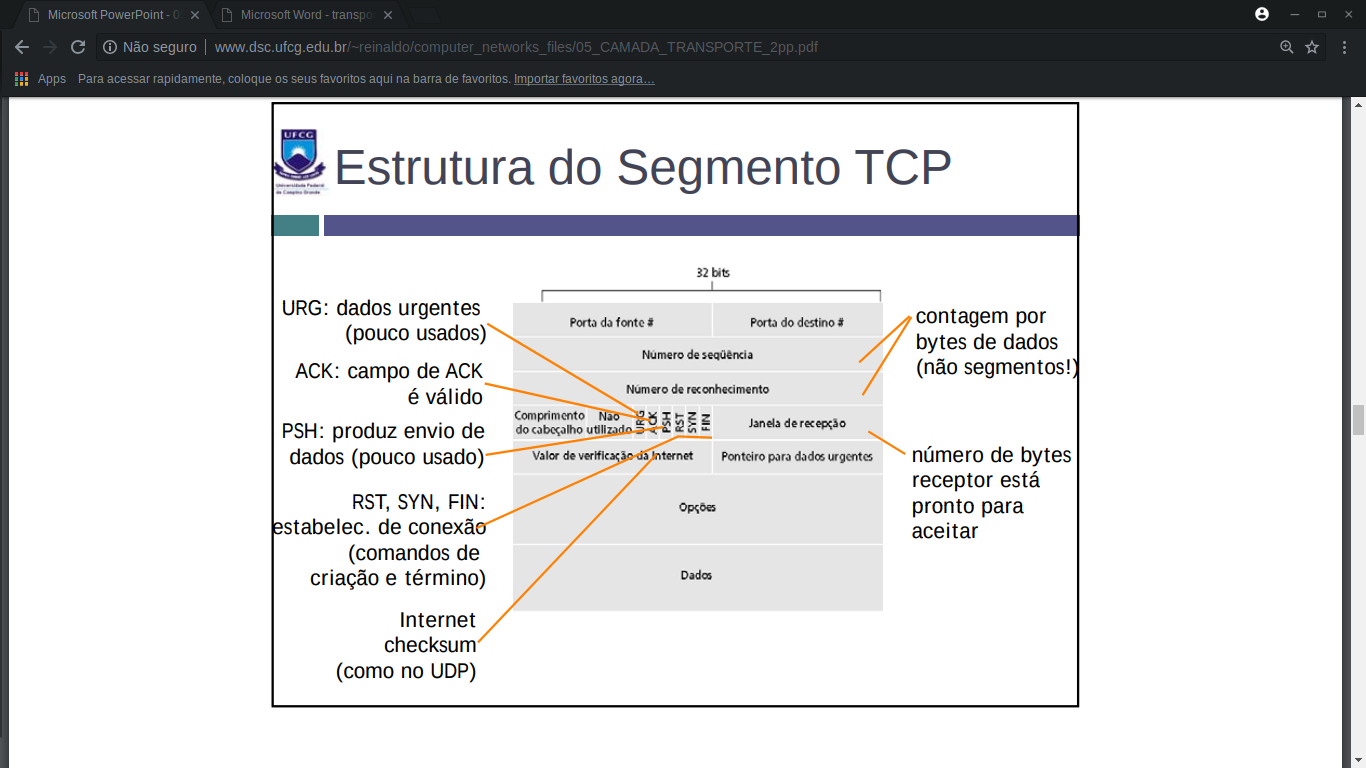
**O TCP é Full-duplex** - É possível a transferência simultânea nas duas direções durante a sessão.

Utiliza o conceito de stream – O TCP envia uma **sequência limitada e contínua de bytes** sem noção dos registros ou quantidade de pacotes que serão recebidos.

Enxerga a rede como uma **conexão ponto-a-ponto** – O protocolo TCP fornece uma conexão diretamente entre aplicativos dos hosts. Tal conexão é denominada conexão virtual ponto-a-ponto, entre o transmissor e receptor, os dispositivos de rede (roteadores) não enxergam a camada de transporte.

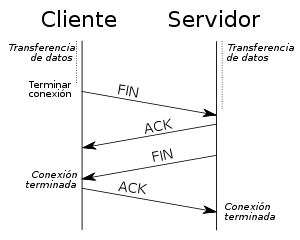
O TCP permite a camada de aplicação enxergar a rede como uma conexão virtual.

Estrutura do Segmento TCP



Número de Sequência e ACKs no TCP

* Números de seqüência:
  + Número do primeiro byte nos segmentos de dados
* ACKs:
  + Número do próximo byte esperado do outro lado
  + ACK cumulativo



1. **Como o receptor trata segmentos fora de ordem?**

*A especificação do TCP não define, fica a critério do implementador.*

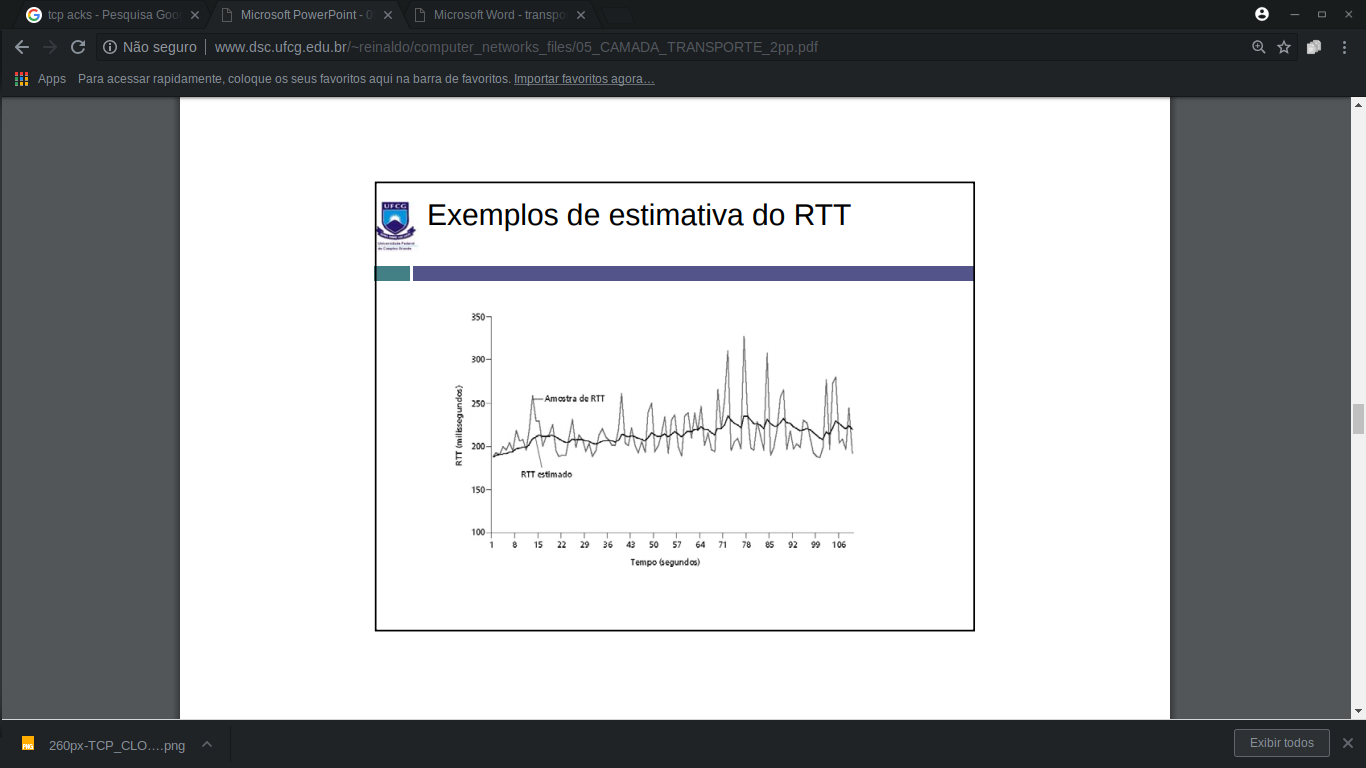
TCP Round Trip Time e temporização

1. **Como escolher o valor da temporização do TCP?**

* ***Maior que o RTT***
  + *Nota: RTT varia*
* ***Muito curto:*** *temporização prematura*
  + *Retransmissões desnecessárias*
* ***Muito longo:*** *a reação à perda de segmento fica lenta*

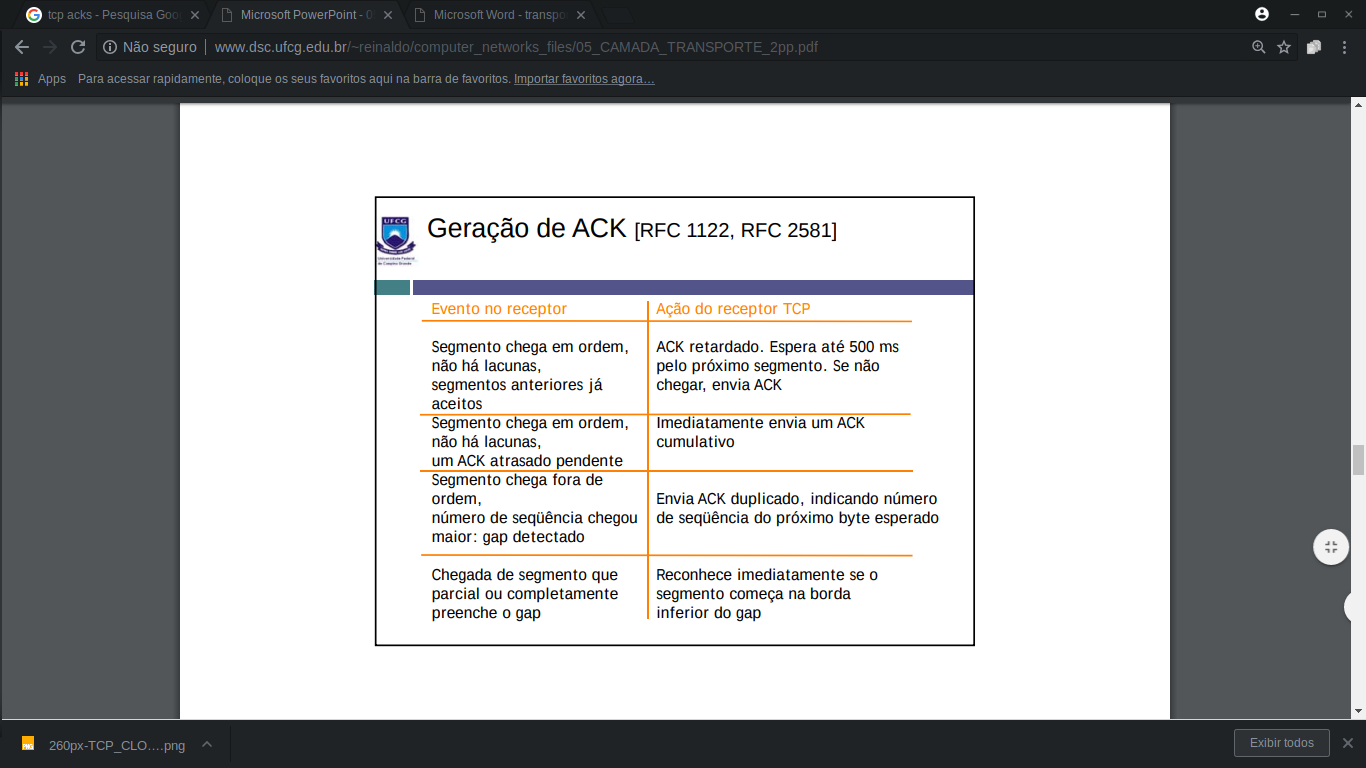
1. **Como estimar o RTT?**

* ***SampleRTT:*** *tempo medido da transmissão de um segmento até a respectiva confirmação*
  + *Ignora retransmissões e segmentos reconhecidos de forma cumulativa*
* *SampleRTT varia de forma rápida, é desejável um* ***amortecedor*** *para a estimativa do RTT*
  + *Usar várias medidas recentes, não apenas o último SampleRTT obtido*

**

TCP: transferência de dados confiável

* TCP cria serviços de transferência confiável de dados em cima do serviço não-confiável do IP TCP: serviço não-confiável do IP.
* Transmissão de vários segmentos em paralelo (Pipelined segments)
* ACKs cumulativos
* TCP usa tempo de retransmissão simples
  + Retransmissões são disparadas por:
    - Eventos de tempo de confirmação
    - ACKs duplicados
* Geração de ACK



Retransmissão rápida

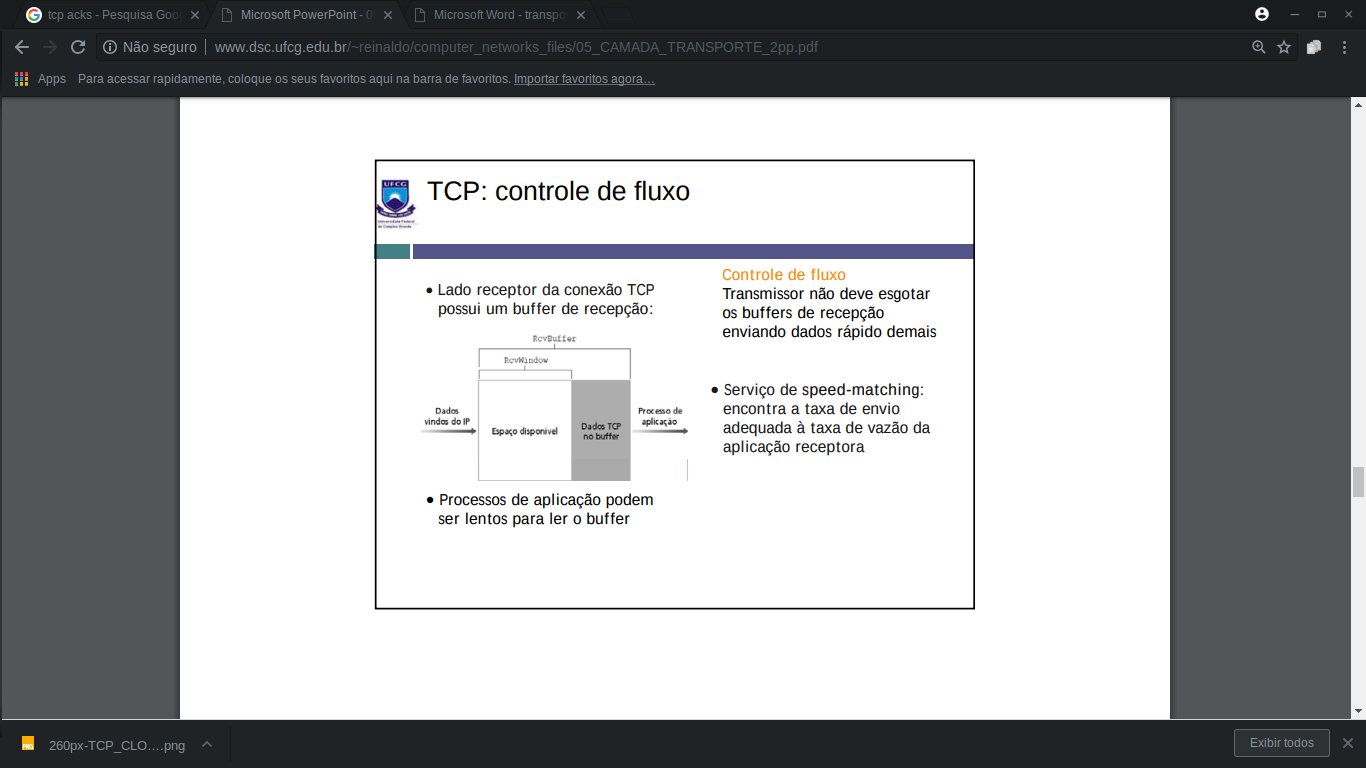
* Com freqüência, o tempo de expiração é relativamente longo:
  + Longo atraso antes de reenviar um pacote perdido
* Detecta segmentos perdidos por meio de ACKs duplicados
  + Transmissor freqüentemente envia muitos segmentos
  + Se o segmento é perdido, haverá muitos ACKs duplicados
* Se o transmissor recebe 3 ACKs para o mesmo dado, ele supõe que o segmento após o dado confirmado foi perdido:
  + Retransmissão rápida: reenvia o segmento antes de o temporizador expirar

TCP: controle de fluxo

Controle de Fluxo (buffers e janelas de transmissão) – Um problema no mundo das redes é garantir o controle de fluxo entre usuários finais. A imprevisibilidade do tráfego é o maior problema. Imagine o resultado do ENEM publicado na internet. Diversos usuários irão fazer requisições em pouco tempo podendo ser mais rápido do que a entrega do servidor web. Assim diversas requisições serão novamente realizadas, gerando ainda mais tráfego e pacotes duplicados. Daí o TCP utiliza o conceito de buffers (armazenamento de pedidos e respostas) e janelas deslizantes:

Janela deslizante é uma característica de alguns protocolos que permite que o remetente transmita mais que um pacote de dados antes de receber uma confirmação. Depois de recebê-lo para o primeiro pacote enviado, o remetente desliza a janela do pacote e manda outra confirmação. O número de pacotes transmitidos sem confirmação é conhecido como o tamanho da janela; aumentando o tamanho da janela melhora-se a vazão.

O receptor da conexão TCP possui um buffer de recepção:



* Receptor informa a área disponível incluindo valor RcvWindow nos segmentos.
* Transmissor limita os dados não confinados ao RcvWindow.
* Garantia contra overflow no buffer do receptor.

Processos de aplicação podem ser lentos para ler o buffer.

O transmissor não deve esgotar os buffers de recepção enviando dados rápido demais. Serviço de speed-matching: encontra a taxa de envio adequada à taxa de vazão da aplicação receptora.

Gerenciamento de Conexão

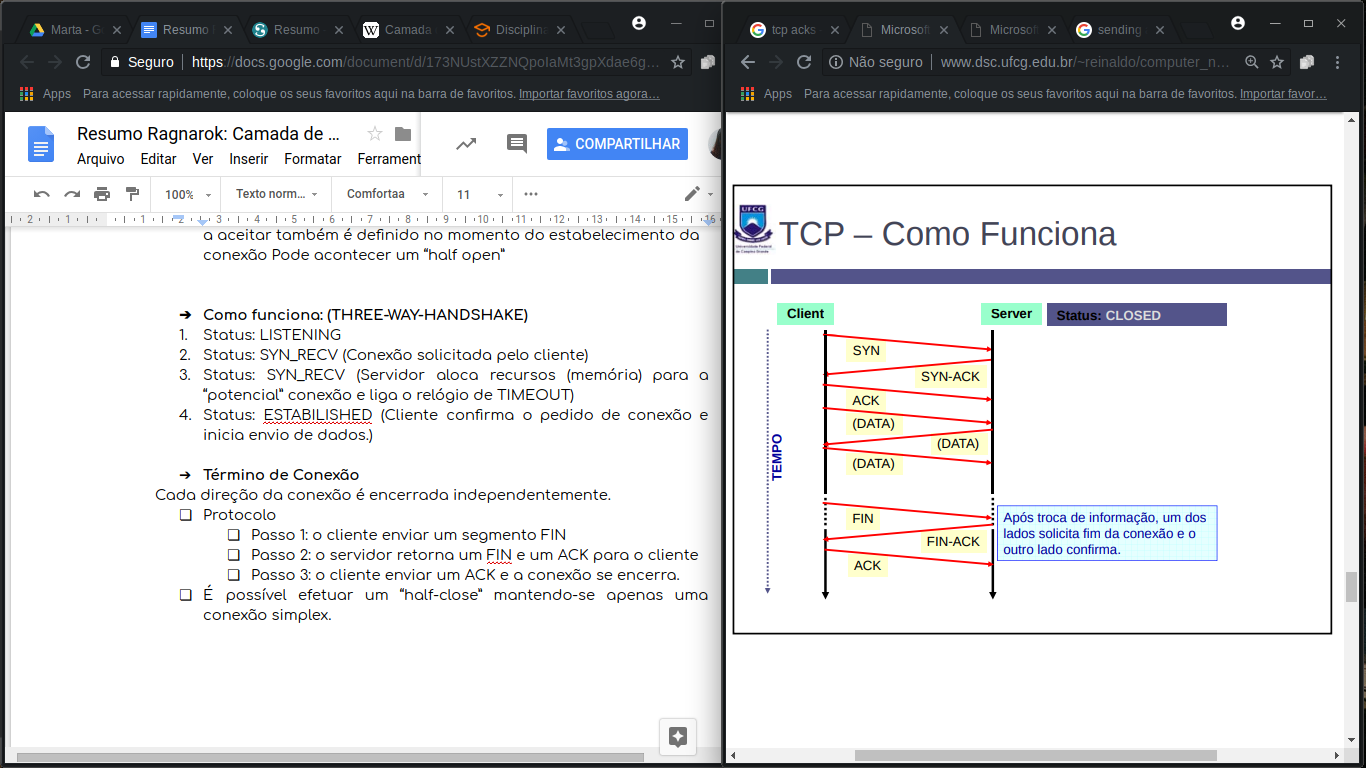
* **Estabelecimento de Conexão**
* Protocolo
  + Passo 1: o cliente envia um segmento SYN especificando a porta do servidor ao qual deseja se conectar e seu número de sequência inicial
  + Passo 2: o servidor responde enviando outro segmento SYN com o ACK do segmento recebido e o seu próprio número de sequência
  + Passo 3: o cliente retorna um ACK e a conexão se estabelece
* O tamanho máximo de segmento (MSS) que cada lado se propõe a aceitar também é definido no momento do estabelecimento da conexão Pode acontecer um “half open”
* **Como funciona: (THREE-WAY-HANDSHAKE)**

1. Status: LISTENING
2. Status: SYN\_RECV (Conexão solicitada pelo cliente)
3. Status: SYN\_RECV (Servidor aloca recursos (memória) para a “potencial” conexão e liga o relógio de TIMEOUT)
4. Status: ESTABILISHED (Cliente confirma o pedido de conexão e inicia envio de dados.)

* **Término de Conexão**

Cada direção da conexão é encerrada independentemente.

* Protocolo
  + Passo 1: o cliente enviar um segmento FIN
  + Passo 2: o servidor retorna um FIN e um ACK para o cliente
  + Passo 3: o cliente enviar um ACK e a conexão se encerra.
* É possível efetuar um “half-close” mantendo-se apenas uma conexão simplex.



Controle de congestionamento do TCP

O controle é feito através de duas variáveis adicionadas em cada lado da conexão: adicionadas em cada lado da conexão:

* **Janela de Congestionamento**
* **Limiar**
  + Serve para controlar o crescimento da janela de congestionamento

Janela de Congestionamento

* Uma conexão TCP controla sua taxa de transmissão limitando o seu número de segmentos que podem ser limitando o seu número de segmentos que podem ser transmitidos sem que uma confirmação seja recebida
  + Esse número é chamado o tamanho da janela do TCP (w)
* Uma conexão TCP começa com um pequeno valor de w e então o incrementa arriscando que exista mais largura de banda disponível
* Isso continua a ocorrer até que algum segmento seja perdido
* Nesse momento, a conexão TCP reduz w para um valor seguro, e então continua a arriscar o crescimento

Janela do Receptor

O número máximo de segmentos não-confirmados é dado pelo mínimo entre os tamanhos das janelas é dado pelo mínimo entre os tamanhos das janelas de congestionamento e do receptor. Ou seja, mesmo que haja mais largura de banda, o receptor também pode ser um gargalo.

Evolução de uma Conexão TCP

No início, a janela de congestionamento tem o tamanho de um segmento. Tal segmento tem o tamanho do maior segmento suportado.

O primeiro segmento é enviado e então é esperado seu reconhecimento.

* Se o mesmo chegar antes que ocorra o timeout, o transmissor duplica o tamanho da janela de congestionamento e envia dois segmentos.
* Se esses dois segmentos também forem reconhecidos antes de seus timeouts, o transmissor duplica novamente sua janela, enviando agora quatro segmentos.

Esse processo continua até que:

* O tamanho da janela de congestionamento seja maior que o limiar, ou maior que o tamanho da janela do receptor;
* Ocorra algum timeouts antes da confirmação.

Duas Fases dessa Evolução

A primeira fase, em que a janela de congestionamento cresce exponencialmente é congestionamento cresce exponencialmente é chamada de inicialização lenta (slow start), pelo fato de começar com um segmento - A taxa de transmissão começa pequena porém cresce muito rapidamente.

Uma vez ultrapassado o limiar, e a janela do receptor ainda não seja um limitante o crescimento receptor ainda não seja um limitante, o crescimento da janela passa a ser linear. Essa segunda fase é chamada de prevenção de congestionamento (congestion avoidance). Sua duração também depende da não ocorrência timeouts, e da aceitação do fluxo por parte do receptor.

E quando ocorrer um problema?

**Na ocorrência de um timeout o TCP irá configurar:**

O valor do limiar passa a ser a metade do tamanho atual da janela de congestionamento

O tamanho da janela de congestionamento volta ser do tamanho de um segmento

O tamanho da janela de congestionamento volta a crescer exponencialmente

**Caso ocorram ocorram 3 ACKs duplicados duplicados:**

O valor do limiar é ajustado para metade tamanho atual da janela de congestionamento

O tamanho da janela de congestionamento passa igual ao valor do limiar (metade da janela de congestionamento atual)

O tamanho da janela de congestionamento cresce linearmente

Resumo

Quando o tamanho da janela de congestionamento está abaixo do limiar, seu crescimento é exponencial.

Quando este tamanho está acima do limiar, o crescimento é linear.

Todas as vezes que ocorrer um timeout, o limiar é modificado para a metade do tamanho da janela e o tamanho da janela passa a ser 1.

A rede não consegue entregar nenhum dos pacotes (“congestionamento pesado”).

Quando ocorrem ACKs repetidos a janela cai pela metade.

A rede ainda é capaz de entregar alguns pacotes (“congestionamento leve”).