Laboratório: Entendendo o Protocolo Stop-and-Wait e a Confiabilidade da Transmissão

Disciplina: Redes de Computadores II

Professora: Angelita Rettore de Araujo

Introdução: Da Teoria do Canal Ideal à Realidade dos Erros

Em laboratórios anteriores, exploramos o conceito de um canal de comunicação ideal, onde a transmissão de dados acontecia sem perdas ou erros. No entanto, no mundo real das redes de computadores, a perfeição é rara. Pacotes podem ser corrompidos (erros de bit), perdidos (devido a congestionamento ou falhas no meio físico) ou entregues fora de ordem.

Para garantir que a comunicação seja confiável, mesmo em canais não confiáveis, são necessários protocolos de Transferência de Dados Confiável (Reliable Data Transfer - RDT). Conforme discutido no Capítulo 3 do livro "Redes de Computadores e a Internet" de Kurose e Ross, um dos primeiros passos para construir um protocolo RDT é o modelo **RDT 2.0**.

O RDT 2.0 aborda o problema de erros de bit introduzindo os seguintes mecanismos:

- **Checksums:** O emissor inclui um campo de checksum (soma de verificação) no pacote para permitir ao receptor detectar erros de bit.
- **Feedback do Receptor:** O receptor precisa informar ao emissor se o pacote chegou corretamente ou não. Isso é feito através de:
    - **ACK (Acknowledgement - Confirmação Positiva):** Indica que o pacote foi recebido sem erros.
    - **NACK (Negative Acknowledgement - Confirmação Negativa):** Indica que o pacote foi recebido com erros.
- **Retransmissão:** Se o emissor recebe um NACK, ele retransmite o último pacote enviado.

Este mecanismo de "enviar e esperar" é a base do que conhecemos como protocolo **Stop-and-Wait**. Nele, o emissor envia um único pacote e aguarda uma confirmação (ACK) antes de enviar o próximo. É um protocolo simples, mas que nos permite entender os desafios fundamentais da confiabilidade.

Neste laboratório, você explorará o protocolo Stop-and-Wait em diferentes cenários, utilizando simulações no NS2, para compreender como ele lida com a confiabilidade e quais são suas limitações.

Objetivos

1. Compreender o funcionamento básico do protocolo Stop-and-Wait em um canal confiável.
2. Analisar como o Stop-and-Wait lida com a perda de confirmações (ACKs).
3. Observar o comportamento do Stop-and-Wait em ambientes com tráfego concorrente.
4. Identificar as limitações do Stop-and-Wait em condições de congestionamento.
5. Aprofundar a compreensão sobre os mecanismos de Transferência de Dados Confiável (RDT).

Material Fornecido

- Scripts TCL para simulações do protocolo Stop-and-Wait:

    - `01-stop-n-wait.tcl`
    - `02-stop-n-wait-loss.tcl`
    - `03-stop-n-wait.tcl`
    - `04-stop-n-wait-loss.tcl`
- Ferramentas para análise de tráfego de rede: `NAM (Network Animator)`.

Simulação 01: Stop-and-Wait em Canal Perfeitamente Confiável

Cenário e Funcionamento

Nesta primeira simulação, você observará o comportamento fundamental do protocolo Stop-and-Wait em um ambiente ideal, ou seja, sem perdas de pacotes ou ACKs e sem erros de bit. O objetivo é visualizar o ciclo básico de **envio-espera-recebimento de ACK-envio do próximo** que caracteriza este protocolo. O emissor (`Emissor-FTP`) enviará pacotes para o receptor (`Receptor-FTP`) e aguardará a confirmação de cada um.

Passo 1: Configuração e Execução

1. O script `01-stop-n-wait.tcl` simula o Stop-and-Wait em um canal simples e confiável.
2. Execute o script utilizando os comandos abaixo no seu Terminal.

Execução no Linux

```
xhost +
docker run --rm -it -e DISPLAY=$DISPLAY -v $PWD:/ns2 -v /tmp/.X11-
unix:/tmp/.X11-unix gelirettore/ns2
ns 01-stop-n-wait.tcl
```

Execução no macOS

Obs: Certifique-se de que o **XQuartz** esteja instalado e configurado no seu macOS para permitir conexões de rede (`Allow connections from network clients`) e que o comando `xhost +` tenha sido executado no Terminal antes de iniciar o contêiner.

```
xhost +
docker run --rm -it -e DISPLAY=host.docker.internal:0 -v $PWD:/ns2
gelirettore/ns2
ns 01-stop-n-wait.tcl
```

Passo 2: Análise do Comportamento

1. Após a execução, o NS2 gerará um arquivo `.nam` (`01-stop-n-wait.nam`) que será automaticamente aberto pelo NAM. Observe a animação do fluxo de pacotes e ACKs.
2. O script também gera um arquivo de rastreamento (`01-stop-n-wait.tr`). Você pode abri-lo com um editor de texto para uma análise detalhada dos eventos (envio e recebimento de pacotes e ACKs).

Para Refletir e Discutir:

- Descreva o padrão de envio de pacotes e recebimento de ACKs que você observou no NAM. Como ele difere de um fluxo contínuo (como o que você viu em simulações UDP/CBR anteriores)?
- O que a "janela de envio" (definida por `$tcp set window_ 1`) significa neste contexto? Como ela impacta a taxa de envio de dados?
- Qual o papel dos ACKs nesta simulação? Eles são essenciais mesmo em um canal sem perdas e erros? Por quê?
- Se o emissor enviasse um pacote e o receptor não enviasse um ACK, o que você esperaria que acontecesse com o emissor no Stop-and-Wait? Qual mecanismo seria acionado?

Discussão Explicativa: O Ciclo Virtuoso do Stop-and-Wait Ideal

Nesta simulação, observamos o funcionamento mais elementar do protocolo Stop-and-Wait, que é uma implementação de um protocolo de Transferência de Dados Confiável (RDT). O que se destaca é o ciclo rígido de "enviar e esperar".

O emissor (`Emissor-FTP`) envia um pacote de dados para o receptor (`Receptor-FTP`). Uma vez que o pacote é enviado, o emissor **para** e aguarda. Ele não enviará outro pacote até receber uma confirmação positiva (ACK) do receptor para o pacote que acabou de enviar. No NAM, isso é visível pelo padrão de "bolinha azul para a direita, bolinha azul para a esquerda (ACK), bolinha azul para a direita, e assim por diante".

O receptor, ao receber um pacote íntegro, envia imediatamente um ACK de volta para o emissor. Somente após receber esse ACK, o emissor pode avançar e enviar o próximo pacote. Este comportamento é imposto pela `window_` (janela de envio) definida como 1 no agente TCP, que limita o número de pacotes não confirmados em trânsito a apenas um.

Em um canal perfeitamente confiável, como simulado aqui, o Stop-and-Wait garante que todos os pacotes cheguem ao destino e na ordem correta. Cada ACK serve como uma validação de que a etapa anterior foi concluída com sucesso, permitindo que a próxima etapa comece. No entanto, é evidente que a eficiência é baixa, pois o emissor passa a maior parte do tempo ocioso, esperando pelas confirmações.

Simulação 02: Stop-and-Wait e a Perda de Confirmações (ACKs)

Cenário e Funcionamento

Agora, vamos introduzir um desafio mais realista: a perda de um ACK. Em redes reais, pacotes de confirmação também podem ser perdidos. Para o protocolo Stop-and-Wait, a perda de um ACK é um problema grave, pois o emissor não tem como saber se o pacote de dados original chegou ao receptor.

Nesta simulação, o script `02-stop-n-wait-loss.tcl` irá simular artificialmente a perda de um ACK (especificamente, o ACK do `Pacote_3`). Você observará como o protocolo Stop-and-Wait, mesmo em sua simplicidade, consegue se recuperar dessa situação.

Passo 1: Configuração e Execução

1. O script `02-stop-n-wait-loss.tcl` é configurado para simular a perda de um ACK.
2. Execute o script utilizando os comandos abaixo no seu Terminal.

Execução no Linux

```
xhost +
docker run --rm -it -e DISPLAY=$DISPLAY -v $PWD:/ns2 -v /tmp/.X11-
unix:/tmp/.X11-unix gelirettore/ns2
ns 02-stop-n-wait-loss.tcl
```

Execução no macOS

```
xhost +
docker run --rm -it -e DISPLAY=host.docker.internal:0 -v $PWD:/ns2
gelirettore/ns2
ns 02-stop-n-wait-loss.tcl
```

Passo 2: Análise do Comportamento

1. Observe atentamente a animação no NAM, prestando especial atenção ao momento em que o `Pacote_3` é enviado e o que acontece em seguida.
2. No arquivo `.tr`, procure pelos eventos de envio do `Pacote_3`, a ausência do ACK correspondente, e a retransmissão.

Para Refletir e Discutir:

- O que você observou acontecer após o envio do `Pacote_3`? Como o emissor "descobriu" que o ACK foi perdido?
- Qual mecanismo do Stop-and-Wait foi acionado para lidar com a perda do ACK? Descreva a sequência de eventos.
- Por que o emissor retransmite o `Pacote_3`? O receptor já havia recebido o `Pacote_3` na primeira tentativa? Se sim, como o receptor lida com a retransmissão de um pacote que ele já recebeu?
- Qual o impacto da perda de um ACK na eficiência da transmissão? Compare com a Simulação 01.

Discussão Explicativa: A Resposta ao Silêncio - Timeout e Retransmissão

Nesta simulação, o protocolo Stop-and-Wait revela sua capacidade de se recuperar de perdas. Quando o `Pacote_3` é enviado, o receptor o recebe e envia o ACK correspondente. No entanto, o script manipula a fila de forma que **o ACK do Pacote_3 é perdido** no caminho de volta.

Para o emissor, a ausência do ACK significa que ele não recebeu a confirmação de que o `Pacote_3` chegou. Como o Stop-and-Wait opera com uma janela de 1, o emissor simplesmente aguarda indefinidamente? Não! É aqui que entra o mecanismo de **timeout (temporizador)**.

O emissor, ao enviar um pacote, inicia um temporizador. Se o ACK correspondente não for recebido antes que o temporizador expire, o emissor assume que o pacote (ou seu ACK) foi perdido. Ele, então, **retransmite** o último pacote enviado (`Pacote_3` novamente).

O receptor, ao receber a segunda cópia do `Pacote_3`, percebe que se trata de uma duplicata (pois já havia recebido o original). Ele descarta a duplicata e envia outro ACK para o `Pacote_3` original. Este ACK finalmente chega ao emissor, que o processa e pode seguir em frente com o envio do `Pacote_4`.

Embora eficaz em garantir a confiabilidade, a retransmissão por timeout penaliza a eficiência. O emissor desperdiça tempo esperando e depois retransmite dados que, em alguns casos (como a perda de ACK), já haviam sido recebidos pelo destino. Isso demonstra uma das grandes desvantagens do Stop-and-Wait: seu baixo desempenho em redes com atrasos significativos ou alta taxa de perdas.

Simulações 03 e 04: Stop-and-Wait em Ambientes Concorrentes e Congestionados

Cenário e Funcionamento

Até agora, as simulações focaram em um único fluxo Stop-and-Wait. No entanto, redes reais possuem múltiplos fluxos de dados concorrentes. Além disso, o **congestionamento** – um excesso de tráfego que os roteadores não conseguem processar – é um problema comum. Quando roteadores ficam congestionados, suas filas enchem e eles começam a descartar pacotes.

As simulações `03-stop-n-wait.tcl` e `04-stop-n-wait-loss.tcl` apresentam um cenário com dois fluxos concorrentes: um fluxo Stop-and-Wait (FTP, azul) e um fluxo CBR (Constant Bit Rate, vermelho).

- **Simulação 03 (`03-stop-n-wait.tcl`):** O limite das filas dos roteadores é alto (`100 pacotes`), o que minimiza a chance de perdas por congestionamento.
- **Simulação 04 (`04-stop-n-wait-loss.tcl`):** O limite das filas dos roteadores é muito reduzido (`2 pacotes`), forçando o congestionamento e a perda de pacotes.

Passo 1: Configuração e Execução da Simulação 03

1. O script `03-stop-n-wait.tcl` configura dois fluxos em paralelo com filas grandes.
2. Execute o script:

Execução no Linux

```
xhost +
docker run --rm -it -e DISPLAY=$DISPLAY -v $PWD:/ns2 -v /tmp/.X11-
unix:/tmp/.X11-unix gelirettore/ns2
ns 03-stop-n-wait.tcl
```

Execução no macOS

```
xhost +
docker run --rm -it -e DISPLAY=host.docker.internal:0 -v $PWD:/ns2
gelirettore/ns2
ns 03-stop-n-wait.tcl
```

Passo 2: Análise do Comportamento da Simulação 03

1. Observe o NAM. Como os dois fluxos (azul e vermelho) se comportam? Há alguma interferência aparente?

2. Qual a taxa de envio de pacotes em cada fluxo? Ambos operam de forma contínua ou um é mais "intermitente"?

Para Refletir e Discutir (Simulação 03):

- Como o fluxo CBR (vermelho) se comporta em comparação com o fluxo FTP (azul)? Qual deles parece transmitir dados de forma mais constante? Por quê?
- Neste cenário com filas amplas, o Stop-and-Wait (FTP) consegue manter uma transmissão relativamente estável? Quais características do protocolo contribuem para isso, mesmo com o tráfego concorrente?

Discussão Explicativa: Concorrência sem Congestionamento Severo (Simulação 03)

Na Simulação 03, você observou a coexistência do fluxo Stop-and-Wait (FTP) e do fluxo CBR em uma rede com capacidade suficiente para ambos (filas grandes). O fluxo CBR é um fluxo de taxa constante; ele envia pacotes sem se preocupar com ACKs ou com a capacidade da rede, preenchendo o link de forma agressiva.

O Stop-and-Wait, por sua vez, mantém seu comportamento de "enviar e esperar". Mesmo com o tráfego CBR concorrente, a largura de banda e o tamanho das filas são suficientes para que os pacotes Stop-and-Wait e seus respectivos ACKs não sejam perdidos. O Stop-and-Wait continua a garantir a entrega confiável, embora sua eficiência intrínseca (apenas um pacote por vez) não seja aprimorada pela presença do outro fluxo.

Este cenário serve para mostrar que, sem gargalos severos, diferentes tipos de tráfego podem coexistir, e o Stop-and-Wait ainda opera de forma confiável, realizando seu trabalho de "garantir a entrega".

Passo 3: Configuração e Execução da Simulação 04

1. O script `04-stop-n-wait-loss.tcl` reduz drasticamente o limite da fila dos roteadores (`2 pacotes`), simulando congestionamento.
2. Execute o script:

Execução no Linux

```
xhost +
docker run --rm -it -e DISPLAY=$DISPLAY -v $PWD:/ns2 -v /tmp/.X11-
unix:/tmp/.X11-unix gelirettore/ns2
ns 04-stop-n-wait-loss.tcl
```

Execução no macOS

```
xhost +
docker run --rm -it -e DISPLAY=host.docker.internal:0 -v $PWD:/ns2
gelirettore/ns2
ns 04-stop-n-wait-loss.tcl
```

Passo 4: Análise do Comportamento da Simulação 04

1. Observe o NAM. Qual o impacto da redução do `queue-limit` nos dois fluxos?
2. Você consegue identificar perdas de pacotes (tanto de dados quanto de ACKs) e subsequentes retransmissões no fluxo Stop-and-Wait?
3. Compare a taxa de transferência do fluxo FTP (azul) nesta simulação com a Simulação 03. O que mudou?

Para Refletir e Discutir (Simulação 04):

- Qual o principal sintoma do congestionamento observado no NAM? Como isso afeta a entrega dos pacotes?
- Como o Stop-and-Wait reage a esse cenário de congestionamento? Ele consegue manter a confiabilidade? E a eficiência?
- Qual o papel das anotações no NAM (`"Pacote_1 perdido (devido à fila cheia)"`) para entender o que está acontecendo?
- Considerando a forma como o Stop-and-Wait opera, qual é a principal razão para sua baixa eficiência em redes congestionadas?

Discussão Explicativa: O Ponto Fraco do Stop-and-Wait - Luta Contra o Congestionamento

A Simulação 04 é um exemplo claro do calcanhar de Aquiles do protocolo Stop-and-Wait em redes reais. Ao reduzir o `queue-limit` dos roteadores para apenas 2 pacotes, forçamos um cenário de **congestionamento severo**. O tráfego CBR (vermelho), por ser insensível ao congestionamento, continua a injetar pacotes na rede, rapidamente preenchendo as pequenas filas dos roteadores.

Isso leva a perdas de pacotes: tanto os pacotes de dados do fluxo FTP (azul) quanto seus ACKs de retorno são descartados pelos roteadores quando as filas estão cheias. Como vimos na Simulação 02, a perda de um ACK ou de um pacote de dados força o Stop-and-Wait a acionar seu temporizador e retransmitir.

No NAM, você pode ver que o fluxo azul se torna ainda mais intermitente. O emissor FTP passa muito tempo em estado de espera, esperando por ACKs que foram perdidos, e realizando retransmissões. Cada retransmissão significa que o emissor está usando recursos da rede para enviar dados que ou já chegaram ou nunca deveriam ter sido enviados novamente, caso um mecanismo mais inteligente de controle de congestionamento estivesse em ação.

A confiabilidade é mantida (eventualmente os pacotes chegam), mas a **eficiência e a taxa de transferência (throughput)** do Stop-and-Wait caem drasticamente em um ambiente congestionado. Ele não possui mecanismos para **sondar a capacidade da rede** ou **diminuir sua taxa de envio** quando detecta congestionamento. Sua resposta fixa (enviar um de cada vez e retransmitir se não houver ACK) não é adequada para cenários dinâmicos e congestionados, levando a um uso ineficiente da largura de banda e a um desempenho pobre.

Reflexão Final

Através dessas simulações, você pôde observar as características fundamentais do protocolo Stop-and-Wait. Ele é um protocolo de Transferência de Dados Confiável capaz de garantir a entrega de pacotes, mesmo em canais com perdas e erros, utilizando checksums, ACKs/NACKs e retransmissões por timeout.

No entanto, sua simplicidade tem um custo. A restrição de ter apenas um pacote "em voo" por vez o torna extremamente ineficiente, especialmente em redes com atrasos significativos (como a Internet) ou que

sofrem de congestionamento frequente. Essa baixa eficiência é a principal motivação para o desenvolvimento de protocolos de transporte mais avançados, como os baseados em Janela Deslizante (Sliding Window), que você explorará em aulas futuras.

Pense nos desafios da Internet em grande escala. Como um protocolo como o Stop-and-Wait se sairia, por exemplo, na transmissão de um arquivo grande para um servidor do outro lado do mundo, ou em uma rede Wi-Fi congestionada? Suas observações neste laboratório devem fornecer uma base sólida para apreciar a engenhosidade e a complexidade dos protocolos de transporte que permitem a comunicação eficiente e confiável no cenário global da rede.
