Redes de Computadores I: Especificação do Trabalho

Prof. Diego Passos, Universidade Federal Fluminense 1/2016

Descrição e Objetivo

O trabalho consiste na implementação de uma versão simplificada do TCP na camada de aplicação sobre um *socket* UDP. Na verdade, trata-se de um pequeno programa de transferência de arquivo sobre *socket* UDP que implementa mecanismos de transferência confiável de dados do TCP.

A aplicação deve ser estruturada seguindo a arquitetura Cliente–Servidor. O lado servidor simplesmente abrirá um *socket* UDP e esperará por conexões do cliente. Ao receber uma conexão, o lado servidor deverá criar um novo arquivo no sistema de arquivos locais, receber o fluxo de bytes enviado pelo cliente e armazená-lo no arquivo recém-criado. O nome do arquivo a ser criado no lado servidor deverá ser lido a partir do usuário (a forma de leitura não é importante, podendo ser, por exemplo, através de parâmetros de linha de comando). Cada execução do lado servidor da aplicação deve corresponder ao recebimento de um único arquivo. Em outras palavras, após o recebimento de todos os bytes do arquivo, o servidor deverá fechar a conexão, fechar o arquivo recém-criado e encerrar sua execução.

Já o lado cliente deverá deverá receber como entradas do usuário o nome do arquivo a ser transferido para o servidor e o endereço do servidor (novamente, a maneira pela qual o programa obtém estas informações não é importante). O cliente deverá, então, abrir o arquivo para leitura a partir do sistema de arquivos local e abrir um *socket* UDP para conexão com o servidor. Deste ponto em diante, o cliente deverá ler blocos de bytes do arquivo, empacotá-los em um cabeçalho e realizar o envio dos pacotes de maneira confiável para o servidor sobre o *socket* UDP. Uma vez que o servidor tenha recebido todos o conteúdo do arquivo, o cliente deve encerrar sua execução.

As duas partes do programa – cliente e servidor – podem ser implementadas na forma de um único programa executável ou como dois programas separados. O *socket* do lado servidor deve ser aberto em uma porta fixa, **definida como uma constante no código-fonte** (tanto no lado servidor, quanto no lado cliente). Sugere-se a utilização da porta 8002. O número de bytes de cada bloco lido do arquivo de origem e enviado em um pacote pelo cliente ao servidor **também deve ser definido como uma constante no código-fonte**. Sugere-se o uso de blocos de 1000 bytes.

Os seguintes mecanismos do TCP devem ser utilizados nesta implementação:

- Número de Sequência – Cada pacote enviado do cliente para o servidor deverá ter um número de sequência informado em seu cabeçalho. Assim como no TCP, **os números de sequência devem ser contados em bytes, e não em pacotes**. O número de sequência

enviado no cabeçalho de um pacote de dados **corresponde ao número de sequência do primeiro byte no pacote**. Ao contrário do que ocorre no TCP, no entanto, **o número de sequência inicial é sempre 0**. O servidor deve usar os números de sequência para garantir a correta ordem dos bytes no arquivo recém-criado. O que o servidor faz com o conteúdo de um pacote fora de ordem (*e.g.*, descartá-lo ou mantê-lo em um *buffer* para uso posterior) é de livre escolha do implementador.

- ACKs – Ao receber pacotes do cliente, o servidor deverá enviar ACKs, reconhecendo a correta recepção do pacote de dados. Assim como no TCP, **os ACKs devem ser cumulativos**. Em seu cabeçalho, **o ACK deverá informar o número de sequência do próximo byte esperado**. Também como feito pelo TCP, no caso de pacotes fora de ordem, o servidor deverá enviar um ACK (possivelmente duplicado) com o número de sequência do próximo byte esperado (em ordem).

- Retransmissões – Ao receber **um ACK duplicado, o cliente deverá reenviar o pacote** de número de sequência indicado no ACK. Note que a retransmissão **deve ser disparada logo na primeira duplicata** do ACK, ao contrário do que o TCP faz.

- Temporizador – Assim como no TCP, o cliente deverá manter um temporizador que determina até quando o cliente deverá esperar pelo ACK de um dado pacote. A manipulação do temporizador é similar à feita pelo TCP. Se, ao transmitir um pacote, não houver temporizador ativo, o mesmo deve ser ativado. Ao receber o ACK não duplicado, o cliente deve desativar o temporizador. Neste caso, se ainda houver outros pacotes em trânsito (*i.e.*, já transmitidos, porém ainda não reconhecidos), o temporizador deve ser ativado novamente com seu valor inicial. Se o temporizador expira, o pacote mais antigo ainda não reconhecido deve ser retransmitido e o temporizador deve ser reativado com seu valor inicial. Ao contrário do que faz o TCP, este valor inicial do temporizador deve ser definido como uma constante no código-fonte.

- Janela de Congestionamento – O cliente deve utilizar uma janela de congestionamento que limita o número de pacotes em trânsito (*i.e.*, transmitidos, porém ainda não reconhecidos). Ao contrário do TCP, **está janela terá comprimento fixo, definido por uma constante no código-fonte**.

O formato dos cabeçalhos dos pacotes de dados (*i.e.*, transmitidos do cliente para o servidor) e dos ACKs (transmitidos do servidor para o cliente) é de livre escolha, **desde que contenham as informações requiridas nesta especificação**. Note que o cliente, de alguma forma, precisa avisar ao servidor quando o arquivo está completo (*i.e.*, qual é o último byte do arquivo). Sugere-se que haja um campo no cabeçalho dos pacotes de dados indicando que aquele é o último bloco. Note ainda que todos os pacotes terão o mesmo tamanho (tamanho do bloco mais cabeçalhos), exceto, potencialmente, pelo último.

Sugere-se, embora não seja obrigatória, a utilização de *threads* separadas no cliente para lidar com a transmissão de novos pacotes e a recepção dos ACKs (e execução das ações

associadas). Neste caso, deve-se ter cuidado com a manipulação de variáveis compartilhadas (como os limites da janela de congestionamento).

O trabalho deverá ser realizado em grupos de até 4 alunos. A linguagem utilizada para a implementação é de livre escolha do grupo, mas a compilação do código deve depender apenas de compiladores/ferramentas gratuitos. A implementação deverá utilizar para a comunicação apenas a API de *sockets* disponível na linguagem, sem outras bibliotecas externas que implementem todos ou parte dos requisitos do programa.

Requisitos

Em resumo, a implementação deverá cumprir os seguintes requisitos:

1. **[Requisito Básico]** Possuir módulo cliente e módulo servidor, seja na forma de um único programa parametrizável, seja na forma de dois programas separados.
2. **[Requisito Básico]** O módulo cliente deverá obter do usuário, de alguma maneira, o nome do arquivo a ser transferido e o nome/endereço IP do servidor.
3. **[Requisito Básico]** O módulo servidor deverá obter do usuário, de alguma maneira, o nome do arquivo a ser criado.
4. **[Requisito Básico]** Ambos os módulos, cliente e servidor, deverão definir em seus códigos-fonte as constantes numéricas especificadas na seção anterior (*i.e.*, número de porta do servidor, tamanho do bloco, tamanho da janela de congestionamento, *timeout* do temporizador) na forma de constantes facilmente alteráveis.
5. **[Requisito de Comunicação]** O módulo cliente deverá ler o arquivo especificado pelo usuário e transmiti-lo através de um *socket* UDP para o servidor.
6. **[Requisito de Comunicação]** O módulo servidor deverá receber o conteúdo do arquivo do cliente e escrever o fluxo de bytes no arquivo especificado pelo usuário.
7. **[Requisito de Confiabilidade]** O programa deverá utilizar números de sequência, conforme descrito na seção anterior.
8. **[Requisito de Confiabilidade]** O programa deverá utilizar ACKs, conforme descrito na seção anterior.
9. **[Requisito de Confiabilidade]** O programa deverá utilizar Retransmissões, conforme descrito na seção anterior.
10. **[Requisito de Confiabilidade]** O programa deverá utilizar Temporizador, conforme descrito na seção anterior.
11. **[Requisito de Confiabilidade]** O programa deverá utilizar Janela de Congestionamento, conforme descrito na seção anterior.

12. **[Requisito de Confiabilidade]** O módulo servidor deverá escrever os bytes em ordem no arquivo de destino, mesmo se pacotes transmitidos pelo cliente chegarem fora de ordem.

13. **[Requisito de Confiabilidade]** O módulo cliente deverá sinalizar ao servidor de alguma forma o final do arquivo. Ambos os módulos deverão encerrar suas execuções quando o arquivo tiver sido completamente transmitido.

Data de Entrega

A data limite para a entrega do trabalho está disponível no calendário da página da disciplina. A entrega deverá ser realizada por e-mail, através do endereço `dpassos@ic.uff.br`. O e-mail deverá conter:

- identificador do trabalho (*e.g.*, "Trabalho de Redes I");
- lista dos integrantes do grupo;
- código fonte da implementação; e
- instruções de compilação/execução/uso da implementação.

Serão aceitos, sem penalidade, e-mails enviados até as 23:59 da data limite. Os e-mails de entrega de trabalho terão seus recebimentos devidamente confirmados. **É responsabilidade do grupo garantir que o trabalho seja recebido, aguardando pela confirmação e reenviando a mensagem caso não a recebam em tempo razoável**.

Em caso de dúvidas ou correções relacionadas a esta especificação, também é responsabilidade de cada grupo entrar em contato (seja pessoalmente, ou através do mesmo endereço de e-mail) requisitando esclarecimentos **dentro do prazo de entrega do trabalho**.

Uma vez entregue o trabalho, não serão aceitas alterações (nem inclusões, nem remoções) na lista de integrantes do grupo em nenhuma hipótese. Por isso, sugere-se atenção no momento do envio da mensagem para que a lista contenha todos os integrantes do grupo.

Critério de Avaliação

Os trabalhos serão avaliados em uma escala de 0 a 10 pontos. A avaliação será dividida nas seguintes partes:

- Aderência aos requisitos básicos (até 0,25 ponto por requisito).
- Aderência aos requisitos comunicação (até 0,5 ponto por requisito).
- Aderência aos requisitos de confiabilidade (até 1 ponto por requisito).
- Existência e qualidade das instruções de compilação/execução/uso da implementação (até 1 pontos).

A cada item avaliado, poderão ser atribuídas frações das pontuações máximas. Trabalhos entregues fora da data serão aceitos, **mas com uma penalidade de 1 ponto por dia (ou fração) de atraso**.

Pontuações extras poderão ser atribuídas ao trabalho caso a implementação contemple as seguintes funcionalidades **opcionais**:

- Implementação do mecanismo de estimativa do *timeout* usado pelo TCP (até 0,5 ponto).
- Implementação de um controle de congestionamento dinâmico baseado ou contendo o *Slow Start* do TCP (até 1 ponto).
- Implementação de um controle de congestionamento dinâmico baseado ou contendo o *Congestion Avoidance* do TCP (até 1 ponto).

Em caso de pontuação extra, mesmo se a nota do trabalho ultrapassar 10,0 pontos, a nota total será utilizada para efeito do cálculo da média da disciplina.