Redes de Computadores

(RCOMP – 2015/2016)

Desenvolvimento de aplicações de rede UDP e TCP

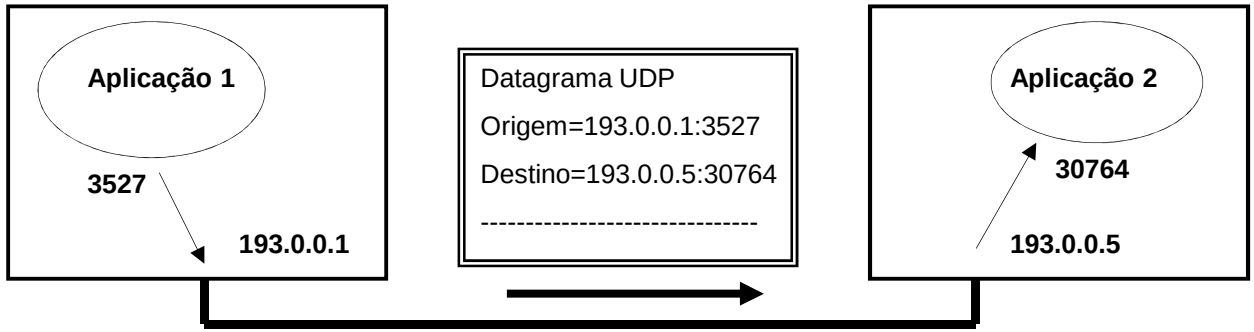
Protocolo UDP (*User Datagram Protocol*)

Tal como o nome indica, trata-se de um serviço de *datagramas*, ou seja permite o envio de blocos de dados de tamanho variável.

Tratando-se de um serviço destinado a ser usado por aplicações de rede, proporciona uma forma de identificação de aplicações individuais.

A origem e destino de cada *datagrama* é identificada por números de 16 bits, conhecidos por números de porto. Os números de porto associados aos endereços IP dos nós identificam universalmente a origem e destino de cada datagrama.

Os números de porto UDP são associados às aplicações através da operação *bind*. Não devem existir duas aplicações no mesmo nó a usar o mesmo número de porto.



Sockets de rede em UDP

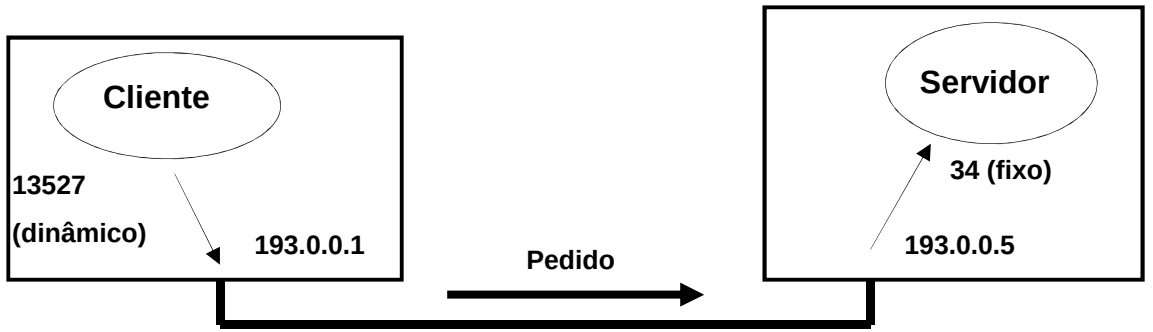
Sob o ponto de vista de desenvolvimento de aplicações, as interações com a rede baseiam-se no conceito de *socket*, no caso do UDP um *socket* é associado a um número de porto local após o que fica em condições de receber e enviar *datagramas* UDP.

A associação de um *socket* UDP a um número de porto (*bind*) é condicionada pela obrigatoriedade de não existir já uma aplicação a usar o mesmo número de porto.

Algumas aplicações não necessitam de usar nenhum número de porto em particular, para estes casos, é normalmente possível solicitar "um qualquer" número de porto que esteja livre. Geralmente isto é conseguido efetuando o *bind* ao número de porto zero.

A maioria das comunicações de rede usa o modelo cliente-servidor, nesse contexto o servidor deve usar um número de porto fixo pré acordado com o cliente. Por outro lado, para o cliente, qualquer número de porto serve (porto dinâmico).

No modelo cliente-servidor o cliente toma a iniciativa de realizar o primeiro contacto, por isso tem de saber antecipadamente o endereço IP do nó do servidor e também o número de porto do servidor.



Envio e receção de *datagramas* UDP

O serviço de *datagramas* UDP é não orientado à conexão e não oferece qualquer tipo de garantias. Cada *datagrama* é tratado individualmente, por isso em cada operação de envio de um *datagrama* é necessário fornecer o endereço IP de destino e o número de porto de destino.

Para cada operação de envio de um *datagrama* deve existir no endereço IP de destino especificado uma aplicação UDP à escuta no número de porto de destino especificado.

Tratando-se de um serviço de *datagramas* é possível enviar em *broadcast*, basta especificar como destino o endereço de *broadcast* de uma rede IPv4, ou 255.255.255.255 para *broadcast* na rede local.

Sob o ponto de vista da aplicação, a receção é normalmente síncrona, ou seja a operação de receção bloqueia a execução da aplicação (processo ou *thread*) até que seja recebido um *datagrama*.

Após a receção de um *datagrama* UDP a aplicação recetora tem acesso ao endereço IP de origem e número de porto de origem. No caso de se tratar de um servidor pode usar estes elementos para enviar mais tarde a resposta ao cliente.

O envio de *datagramas* UDP não oferece qualquer tipo de garantias, nem qualquer *feedback*, ou seja o emissor não tem forma de saber se o *datagrama* chegou ou não ao destino.

Existe uma única exceção a esta falta de *feedback*: quando o nó de destino está operacional, e o número de porto de destino não está em uso por nenhuma aplicação, o nó de destino emite uma mensagem ICMP "Destination port unreachable", no nó de origem a API associa então o erro ao *socket* emissor.

Protocolo TCP (*Transmission Control Protocol*)

O protocolo TCP proporciona um serviço de qualidade significativamente superior ao do protocolo UDP.
O TCP permite criar ligações lógicas bidirecionais entre aplicações residentes em nós de rede distintos. Estas ligações lógicas vulgarmente designadas *conexões TCP* fornecem garantias de entrega dos dados na ordem em que são emitidos.
As ligações TCP são exclusivas dos dois nós entre os quais são criadas, são canais de comunicação dedicados nos quais não é possível a intervenção de terceiros.

Sob o ponto de vista das aplicações, as transações de dados via TCP são realizadas em fluxo, não existe o conceito de bloco de dados, todos os dados são enviados e recebidos byte a byte num fluxo contínuo.

As transações de dados via TCP apenas são possíveis após uma fase prévia de estabelecimento da ligação TCP (*conexão TCP*).

Para ser possível estabelecer uma conexão TCP entre duas aplicações cada uma delas tem de assumir um papel diferente:

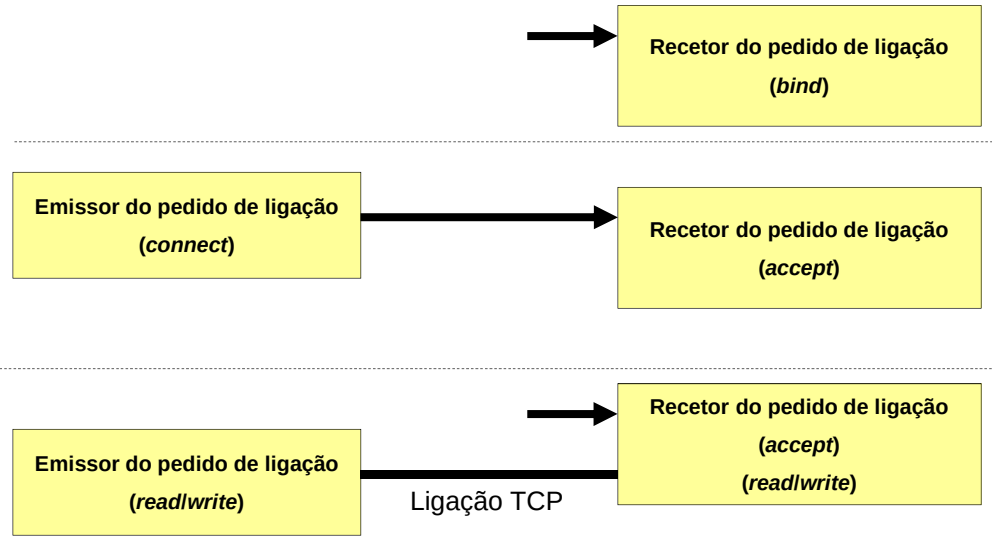
-Uma das aplicações aceita o estabelecimento da ligação TCP (Servidor TCP)

-A outra aplicação pede o estabelecimento da ligação TCP (Cliente TCP)

Estabelecimento da ligação TCP

Para ser possível estabelecer a ligação TCP entre duas aplicações, uma delas assume o papel de recetor do pedido de ligação, para isso associa o *socket* TCP a um número de porto fixo TCP previamente acordado com o emissor do pedido de ligação. De seguida fica à espera.

A outra aplicação pode então emitir um pedido de estabelecimento de ligação TCP especificando como destino o endereço IP do nó onde se encontra a primeira aplicação e o número de porto que essa aplicação está a usar.



A aplicação recetora associa um número de porto TCP ao *socket* TCP (*bind*). A aplicação fica assim disponível para receber pedidos de ligação TCP nesse número de porto.

A aplicação emissora efetua um pedido de ligação (*connect*) tendo como destino o endereço IP do nó recetor e o número de porto usado pela aplicação recetora.

A aplicação recetora aceita o pedido de ligação (*accept*) e a ligação TCP entre as duas aplicações é criada.

Do lado do recetor, a aceitação de um pedido de ligação (*accept*) cria um novo *socket* que corresponde à ligação TCP, o *socket* original continua disponível para receber outros pedidos de ligação.

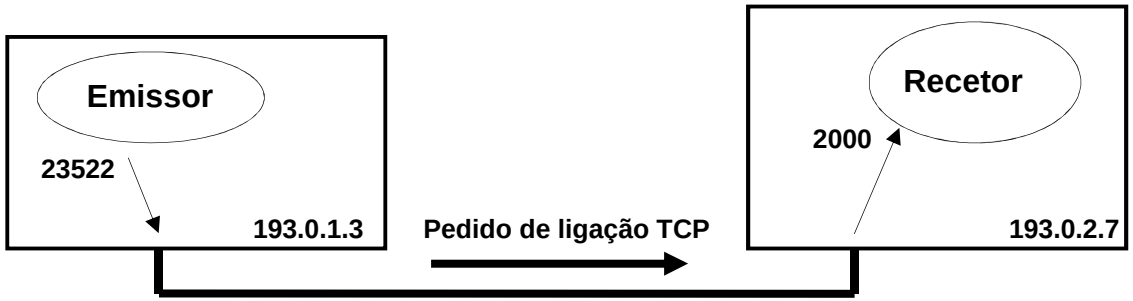
As duas extremidades da ligação TCP são acessíveis, no emissor através do *socket* usado para efetuar o pedido de ligação e no recetor através do novo *socket* criado quando o pedido de ligação foi aceite.

As operações de envio e receção de dados através de ligações TCP são realizadas em fluxo de bytes, normalmente são usadas funções semelhantes às usadas nas operações de leitura e escrita em ficheiros e *pipes*.

Ligações TCP – Canais dedicados

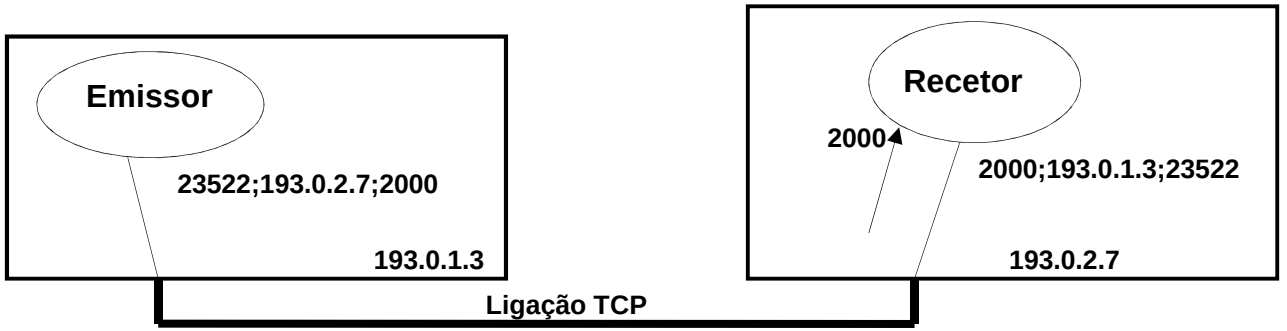
Uma ligação TCP constitui um canal de comunicação dedicado entre duas aplicações, para todos os efeitos pode ser tratado como um *pipe* bidirecional.

Antes de a ligação TCP ser estabelecida, cada um dos *sockets* tem associado a si apenas o número de porto local.



Após o estabelecimento da ligação TCP, os *sockets* correspondentes à ligação têm associados também o endereço IP remoto e o número de porto remoto.

Pode observar-se que no recetor existem *sockets* diferentes associados ao mesmo número de porto local. São distintos porque apenas um deles tem um endereço remoto associado.



Os dados TCP que chegam da rede são disponibilizados no *socket* apenas e só se o número de porto de destino dos dados corresponde ao porto local, o endereço IP remoto corresponde ao endereço de origem dos dados e o número de porto remoto corresponde ao número de porto de origem dos dados.

UDP – Envio e receção de *datagramas*

Uma vez que não se trata de um serviço com ligação, assim que é associado um número de porto local ao *socket* (*bind*) ele encontra-se a disponível para receber *datagramas* da rede e enviar *datagramas*.

Todas as operações de envio e receção de rede são geridas pelo sistema operativo através de filas FIFO. Isto significa que mesmo que a aplicação não solicite explicitamente a receção de um *datagrama*, eles podem estar a ser recebidos e serão disponibilizados à aplicação pela ordem em que chegaram, quando a aplicação o solicitar a sua receção.

O envio de *datagramas* é realizado através da invocação de uma *system-call* que recebe como argumentos um bloco de dados com determinada dimensão e um endereço de destino (endereço IP + número de porto UDP).

A *system-call* que solicita a receção de um datagrama devolve um bloco de dados com determinada dimensão e o endereço de origem do datagrama (endereço IP + número de porto UDP).

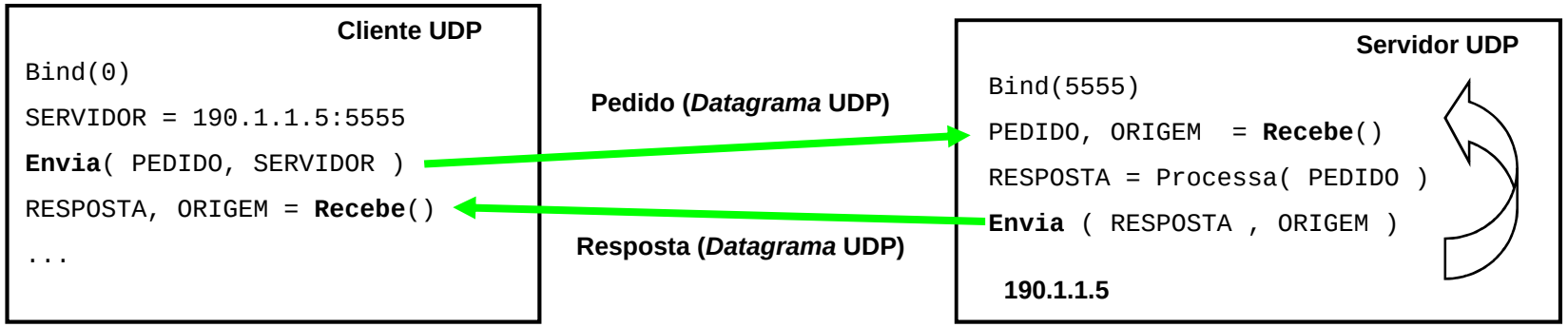
Normalmente as receções de dados são bloqueantes, ou seja, se não existir nenhum *datagrama* na fila, a operação de receção bloqueia o processo/*thread* até que chegue um *datagrama*.

UDP – Serviço não fiável

O serviço UDP é de utilização muito simples, mas não oferece qualquer tipo de garantias, cabe às aplicações (protocolo de aplicação) resolver os problemas que daí possam resultar.

Não havendo qualquer garantia que os *datagramas* emitidos cheguem ao destino, nem sequer qualquer *feedback* relativamente à entrega, as aplicações pouco cuidadosas podem facilmente ficar comprometidas.

A figura seguinte representa a interação normal entre um cliente e um servidor UDP:



Neste caso as consequências da falta de fiabilidade do UDP manifestam-se no cliente que confia que vai chegar uma resposta. Se a resposta não chegar o cliente fica "eternamente" bloqueado na *system-call* Recebe().
Isto pode acontecer por vários motivos: o pedido perdeu-se; o servidor falhou; a resposta perdeu-se.

Clientes UDP – tolerância a falhas

No contexto do modelo cliente-servidor, as limitações do protocolo UDP levam a que cada pedido seja tratado independentemente de pedidos anteriores (servidor idem potente). Sob o ponto de vista de falhas na entrega de *datagramas* o servidor fica numa posição cómoda pois o seu funcionamento nunca será afetado por esse tipo de falhas.

O cliente pelo contrário, depois de enviar o pedido, fica dependente da chegada de uma resposta do servidor. A chave para resolver o problema passa por evitar o bloqueio da aplicação quando fica à espera da receção da resposta definindo um tempo máximo para essa operação (*timeout*). Existem várias formas de implementar esta funcionalidade:

• ***Sockets* não bloqueantes** – é possível alterar o comportamento de um *socket* de forma a tornar-se não bloqueante, depois disso as operações realizadas sobre ele que levariam a um bloqueio retornam de imediato com erro.

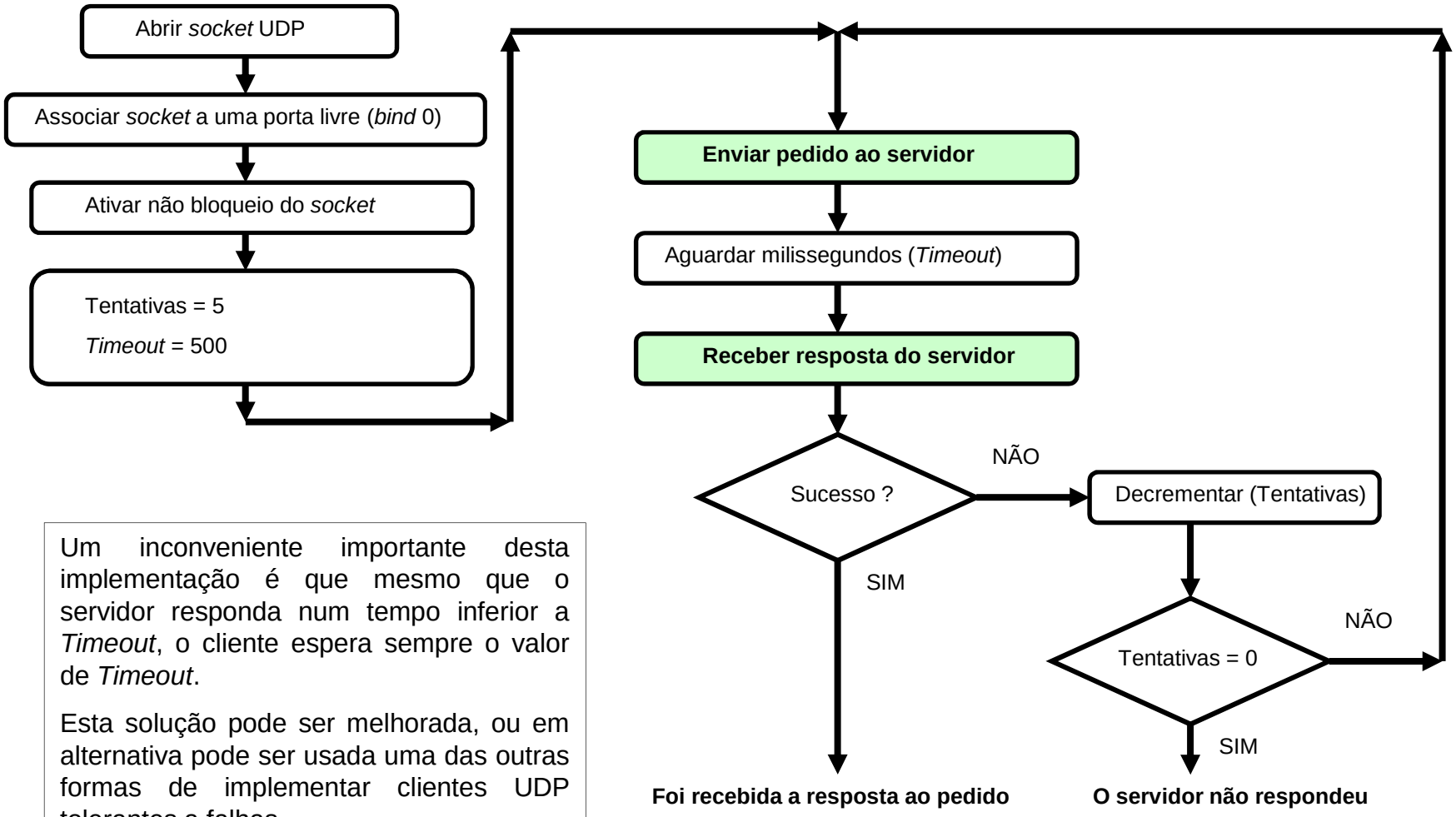
• ***Sockets* com *timeout*** – algumas linguagens (ex.: Java) permitem associar a um *socket* um tempo máximo para as operações sobre ele realizadas, por exemplo se a uma operação de receção demora um tempo superior ao definido será terminada com uma exceção.

• ***Threads*** – uma outra possibilidade é criar um *thread* para efetuar a receção e cancelar o *thread* passado um período de tempo pré-determinado (*timeout*) caso não tenha chegado nenhuma resposta.

• **Monitorização de *sockets*** – algumas API de *sockets* dispõem de funções de monitorização de *sockets*, é o caso da *system-call select* em C, permite por exemplo monitorizar um *socket* UDP para determinar se existe algum *datagrama* disponível para ser recebido. A *system-call select* permite também associar um tempo máximo à operação.

•**Sinais** – em alguns sistemas operativos os processos são avisados da chegada de dados a um *socket* através de sinais, é o caso do sinal SIGIO nos sistemas Unix. Esta característica pode ser aproveitada.

Cliente UDP tolerante a falhas – *Socket* não bloqueante



Um inconveniente importante desta implementação é que mesmo que o servidor responda num tempo inferior a *Timeout*, o cliente espera sempre o valor de *Timeout*.

Esta solução pode ser melhorada, ou em alternativa pode ser usada uma das outras formas de implementar clientes UDP tolerantes a falhas.

UDP – Envio em *broadcast*

Algo que não é possível num serviço com ligação como o TCP, mas é possível num serviço de *datagramas* como o UDP é o envio de dados para um endereço de *broadcast*.

Quando um *datagrama* é enviado para o endereço de *broadcast* todos os nós da rede correspondente vão receber o *datagrama* no número de porto especificado como destino.

Em IPv4 o endereço de *broadcast* de uma rede é o último endereço dessa rede, ou seja o endereço de rede em que todos os bits à direita da mascara de rede têm o valor 1. A colocação no código de uma aplicação de um endereço de *broadcast* nesta forma não é contudo aceitável pois nesse caso a aplicação apenas funcionaria nessa rede.

Para esse efeito o referido endereço de *broadcast* deverá ser colocado num ficheiro de configuração que é lido pela aplicação. Em alternativa, se apenas for pretendido o *broadcast* na rede local, pode ser usado o endereço 255.255.255.255 que permite o envio em *broadcast* na rede local independentemente de qual seja essa rede.

Uma das aplicações mais importantes do envio de *datagramas* UDP em *broadcast* é permitir a um cliente contactar um servidor que se encontra na rede local, mas cujo endereço desconhece.
Esta facilidade é usada em muitos ambientes de rede local como por exemplo Windows/NetBIOS para localizar de forma expedita servidores na “vizinhança da rede”.

UDP – Envio em *broadcast* – localização de aplicações

Mesmo que o protocolo de aplicação utilize TCP, nada impede que numa fase prévia, para efeito de localização das aplicações (descoberta do endereço IPv4) se use UDP em *broadcast*.

Numa arquitetura cliente servidor pode adota-se uma das técnicas seguintes:

Anuncio de servidores – Um servidor pode enviar periodicamente em *broadcast*, um *datagrama* UDP onde anuncia à rede a sua presença. Ao fazer este anuncio dá a conhecer a sua localização (endereço IPv4).
Os clientes deste tipo de aplicação começam por escutar a rede recebendo *datagramas* UDP no porto pré combinado, constroem assim uma lista de servidores disponíveis guardando os respetivos endereços IPv4.

Pedido de servidores – O cliente envia em *broadcast*, um *datagrama* UDP para um número de porto pré combinado, onde solicita servidores.
Os servidores presentes na rede respondem, permitindo igualmente ao cliente a construção de uma lista de servidores disponíveis e respetivos endereços IPv4.

Em qualquer um dos modelos o cliente fica a conhecer uma lista de servidores disponíveis e respetivos endereços IPv4 podendo depois escolher um e comunicar com ele usando UDP ou TCP.

UDP – Tamanho dos *datagramas*

Para além da falta de garantias ou *feedback* da entrega dos *datagramas* UDP, o desenvolvimento de aplicações pode deparar-se com um outro problema:

Que volume de informação pode ser colocado em cada *datagrama* UDP ?

Teoricamente um *datagrama* IPv4 pode ter 65535 bytes, por isso o volume de dados num *datagrama* UDP poderia ter esse valor, descontando o tamanho do cabeçalho IPv4 (20 a 60 bytes) e o tamanho do cabeçalho UDP (8 bytes).

No entanto, atualmente não se usa fragmentação, por outro lado a RFC 791 (IPv4) diz que todos os nós são obrigados a suportar *datagramas* IPv4 com pelo menos 576 bytes.

A forma de garantir que o volume excessivo de dados de um *datagrama* UDP não vai impedir a sua chegada ao destino é evitar ultrapassar 512 bytes (RFC 791).

Quando as transações ultrapassam volumes de 512 bytes só há duas alternativas possíveis:

-Esquecer os *datagramas* UDP e optar antes por conexões TCP.

-Dividir a informação por vários *datagramas* UDP.

UDP – Transações em múltiplos *datagramas*

Normalmente quando se verifica que as transações de um protocolo de aplicação envolvem volumes de dados superiores a 512 bytes, opta-se por usar TCP e não UDP.

Sendo o UDP um protocolo sem ligação, sem qualquer garantia de entrega ou ordem de entrega, a divisão de um bloco de dados por vários *datagramas* UDP vai obrigar as aplicações a realizarem uma série de procedimentos de verificação.

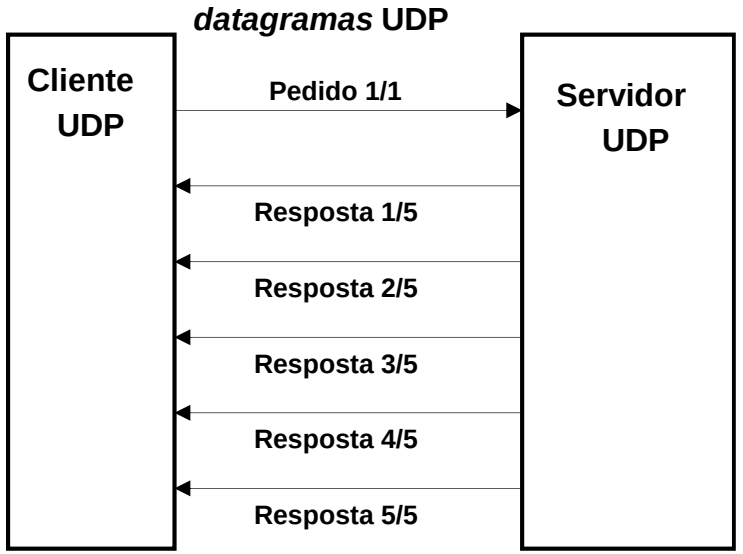
No mínimo, o recetor terá de saber o número total de *datagramas* e estes terão de ser numerados.

O protocolo de aplicação poderá ser mais ou menos refinado em termos de recuperação de erros:

-a falha de entrega de um *datagrama* pode levar à nulidade de toda a transação e levar à sua repetição.

- o cliente, ao detetar a falha de entrega de um *datagrama*, pode solicitar ao servidor o envio apenas do *datagrama* em falta.

Exemplo:



Sockets UDP – associação a endereços remotos

O UDP é um protocolo sem ligação, contudo é possível associar a um *socket* local um endereço remoto através do endereço IP e número de porto remotos.

Na API de linguagem C, a *system-call* usada para este efeito (*connect*) é a mesma que é usada para estabelecer uma ligação TCP, contudo neste caso não se trata de nenhum tipo de ligação . Por exemplo, ao contrário do que acontece com uma ligação TCP, sempre que a aplicação entender pode associar o *socket* a um outro endereço remoto, as vezes que quiser.

A associação de um *socket* UDP a um endereço remoto tem efeito quer sob o ponto de vista de emissão de *datagramas* quer na receção:

EMISSÃO – em cada emissão de um *datagrama* deixa de ser necessário especificar o destino, sendo usado sempre o endereço remoto que foi associado ao *socket*.

RECEÇÃO – o *socket* passa a receber apenas *datagramas* cujo endereço de origem corresponde ao endereço remoto que lhe foi associado.

A associação do *socket* UDP a um endereço remoto não tem mais nenhum efeito, as características de falta de fiabilidade do UDP persistem .

Este tipo de associação pode contudo ser bastante útil em algumas aplicações, em particular a filtragem de *datagramas* que fica associada ao processo de receção. Os servidores UDP *multi-processo* aproveitam esta propriedade.

Ligações TCP – Envio e receção de dados

O protocolo TCP tem a vantagem de garantir a entrega fiável dos dados na ordem exata em que são emitidos, sem qualquer limitação relativamente ao volume de dados enviados ou recebidos. O envio e receção de dados através de uma ligação TCP é realizado byte a byte.

Tem de haver uma correspondência exata entre o número de bytes cuja leitura é solicitada numa extremidade e os número de bytes que foram escritos na outra extremidade.

❖ Se for solicitada a leitura de mais bytes do que os que foram escritos, a leitura vai ficar bloqueada à espera que surjam os bytes em falta. As consequências são obvias.

❖ Se for solicitada a leitura de menos bytes do que os que foram escritos, a leitura conclui-se sem problemas, mas os bytes que não foram lidos vão surgir na leitura seguinte.

A garantia de que esta correspondência existe (sincronização entre leituras e escritas) é da responsabilidade do protocolo de aplicação onde estão definidas todas as trocas de informação entre as entidades envolvidas, tipicamente clientes e servidores.

Ligações TCP - Envio e Receção - Protocolo de aplicação

Um protocolo de aplicação é um conjunto de regras que duas aplicações de rede usam para poderem dialogar sem ambiguidades. Define os procedimentos de cada uma das aplicações e as trocas de informação que têm lugar nas várias fases dos procedimentos.

Quando o protocolo de aplicação usa uma ligação TCP é da sua responsabilidade garantir que vai existir uma correspondência exata entre as operações de escrita e operações de leitura, realizadas nas extremidades opostas da ligação.

Existem três abordagens ao problema que podem ser usadas isoladamente ou combinadas:

- Blocos de tamanho fixo – se o protocolo de aplicação estabelecer mensagens de comprimento fixo em cada situação, a leitura dessas mensagens não apresenta qualquer problema. Esta solução pode conduzir a algum desaproveitamento da rede se o volume de informação útil transmitida for inferior ao tamanho fixo da mensagem.

- Indicador do tamanho do bloco – o emissor começa por informar o recetor do tamanho da mensagem que se segue. Com esse conhecimento o recetor pode solicitar a leitura do número exato de bytes necessário. Exemplo: campo “Content-Length” do protocolo HTTP.

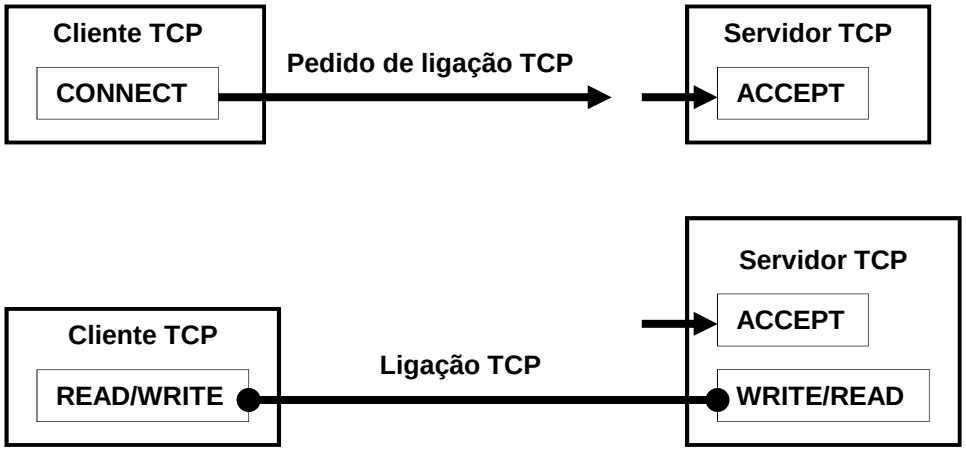
- Marcador de fim de bloco – o protocolo estabelece um marcador para indicar o fim do bloco. O recetor efetua a leitura byte a byte até surgir o marcador. É simples de implementar quando se pode garantir que os dados nunca contêm o marcador escolhido. No cabeçalho das mensagens HTTP (em formato de texto) a sequência CR/LF é usada para identificar o fim das linhas e a sequência CR/LF/CR/LF é usada para identificar o fim do cabeçalho.

Servidores TCP

Um servidor TCP é antes de mais uma aplicação que aceita pedidos de ligação TCP num número de porto definido no protocolo de aplicação.

Quando o servidor TCP aceita um pedido de ligação TCP de um cliente, fica estabelecida uma ligação TCP e do lado do servidor é criado um novo *socket* associado a essa ligação.

A ligação TCP criada entre cliente e servidor constitui um canal de comunicação permanente e dedicado que apenas pode ser usado pelas duas aplicações envolvidas e se mantém disponível até que alguma delas decida fechar a ligação.

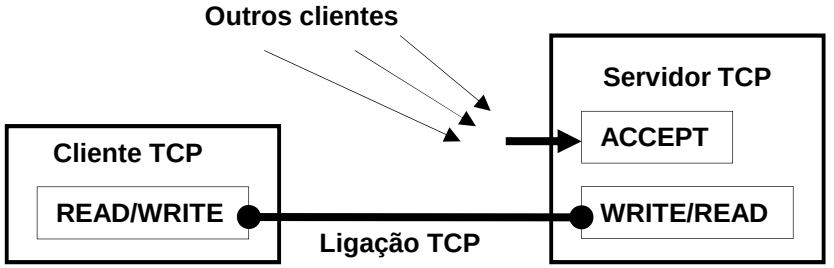


O formato das mensagens trocadas entre cliente e servidor através da ligação TCP é definido pelo protocolo de aplicação, mas graças ao carácter persistente da ligação TCP constitui uma sessão no sentido em que estas interações podem não se limitar apenas a um pedido e respetiva resposta.

Servidores TCP *multi-processo*

Depois de um servidor TCP aceitar uma ligação de um cliente tem de se manter disponível para aceitar outros clientes.

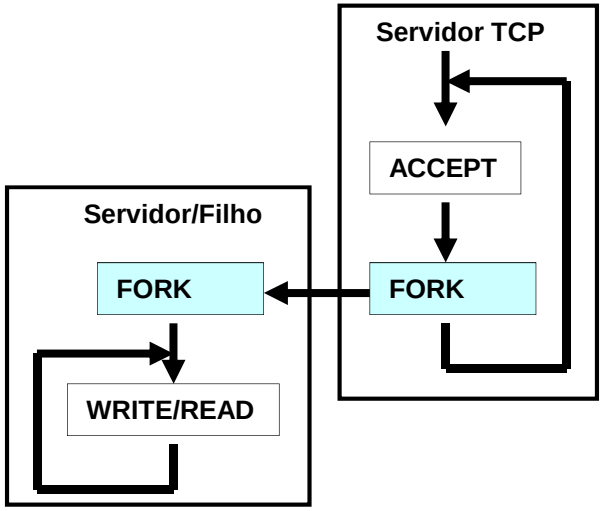
Ou seja, tem de dialogar com o cliente segundo o protocolo de aplicação através do *socket* correspondente à ligação TCP, mas também tem de aceitar novos pedidos de ligação.



Ou seja o servidor tem de lidar com dois *sockets* aceitando novos clientes no *socket* original (*accept*) e dialogando com o cliente no novo *socket* correspondente à ligação TCP.

Existem muitas formas de resolver o problema, em C/Unix uma das mais populares é criar um processo filho exclusivo para o novo cliente.

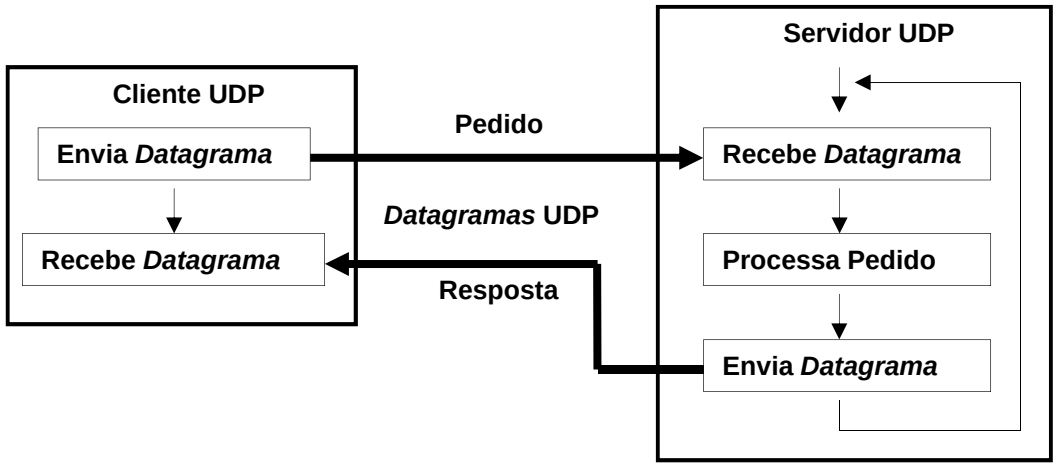
Esta forma de implementar o servidor TCP tem a vantagem de criar um processo independente para atender cada cliente.
Cada um destes processos filho fica inteiramente dedicado a servir um único cliente em particular, com todas as vantagens que daí advêm.



Servidores UDP

Devido às limitações do protocolo UDP, um servidor UDP tem normalmente um funcionamento muito simples, limita-se a receber um pedido no número de porto definido no protocolo de aplicação, sob a forma de um *datagrama*, processar o pedido e enviar a resposta ao cliente.

Dada a ausência de qualquer canal de comunicação dedicado entre cliente e servidor, cada pedido é tratado individualmente. Quando o servidor recebe um pedido guarda o endereço de origem para após o processamento enviar a resposta ao cliente.



Não existe qualquer tipo de sessão, o servidor atende os pedidos pela ordem de chegada, um pedido só é atendido pelo servidor depois de os pedidos anteriores terem sido processados e respondidos.

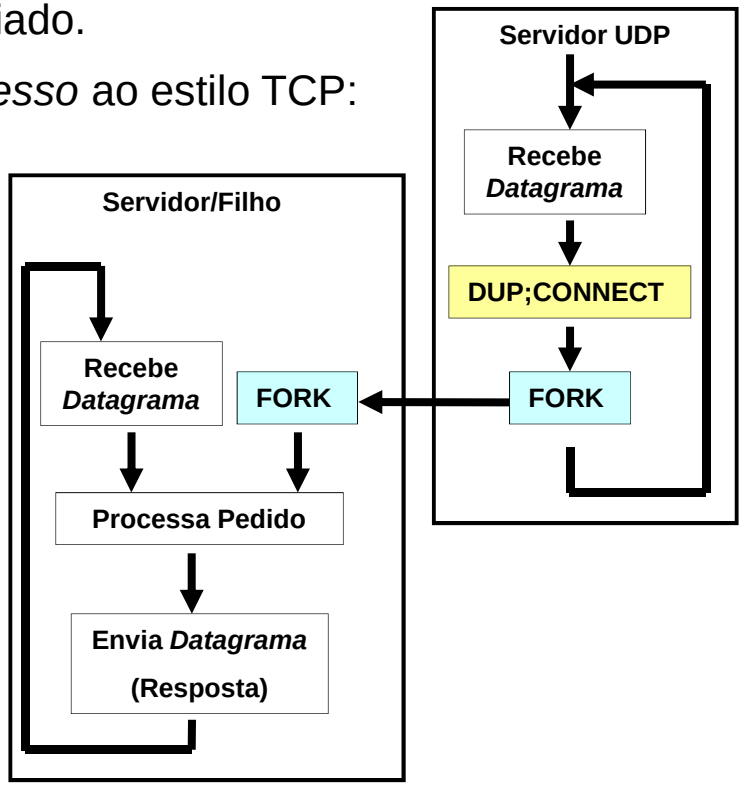
Implementar protocolos de aplicação com sessão sobre UDP não é impossível, mas torna o servidor mais complexo. O servidor terá de armazenar vários contextos de comunicação correspondentes a cada um dos clientes com quem está a comunicar e mediante a chegada de um pedido selecionar o contexto correto tendo com critério o endereço do cliente (endereço + número de porto).

Servidores UDP *multi-processo*

Os servidores UDP normalmente são implementados num único processo, cada porto UDP é um ponto de receção único que não pode ser usado simultaneamente por vários processos sem conflitos. Com vários processos a ler do mesmo *socket* nunca se saberia qual dos processos receberia o *datagrama*.
Contudo através da associação do socket UDP a um endereço remoto obtém-se um ponto distinto de receção pois esse socket passa a receber apenas os *datagramas* que têm como origem o endereço (endereço + porto) que lhe foi associado.

Torna-se então possível uma implementação *multi-processo* ao estilo TCP:

Quando o servidor recebe um pedido cria uma cópia do socket (DUP) e associa a essa cópia o endereço de origem do *datagrama* (CONNECT), de seguida cria um processo filho (FORK), o processo filho usa a cópia que está associada ao endereço remoto. O processo pai continua a usar o socket original que não está associado a nenhum endereço remoto.
Sempre que chega um *datagrama* ao porto UDP, é verificado se o endereço de origem corresponde a um endereço associado a um socket, nesse caso disponibiliza os dados apenas nesse socket.



Protocolo de aplicação

Sob o ponto de vista das redes de computadores, um protocolo é um conjunto de regras que visam permitir a troca de informação sem ambiguidades entre duas entidades que comunicam através de uma infraestrutura de rede. Quando as entidades envolvidas são aplicações finais (situadas no nível 7 – camada de aplicação do MR-OSI) estes protocolos são designados “protocolos de aplicação”.

O protocolo é uma especificação o mais formal e exata possível de:

- Todos os formatos de mensagens usados nas diferentes fases dos diálogos.
- Todas os diálogos e ações possíveis, os respetivos objetivos e resultados possíveis.
- Procedimentos de deteção e recuperação de erros.

O desenvolvimento de um protocolo de aplicação deve ter em consideração os seus objetivos iniciais, como por exemplo que tipos de dados vão ser transferidos.

Os protocolos de aplicação devem ser flexíveis permitindo a implementação de novas funcionalidades mantendo a compatibilidade com versões anteriores:
- O primeiro elemento das mensagens deve identificar a versão do protocolo.
- O formato geral das mensagens deve ser suficiente flexível para comportar novos formatos específicos.

Receção assíncrona

A disponibilidade de dados para serem recebidos da rede constituem eventos assíncronos no sentido em que a aplicação não tem um controlo preciso sobre o instante em que vão ocorrer. Muitas aplicações limita-se a solicitar a leitura e aguardar (bloquear) até que os dados estejam disponíveis, este tipo de procedimento pode designar-se receção síncrona.

Para muitas aplicações esse procedimento não é aceitável, porque:
• estão a usar vários *sockets* e não sabem em qual deles vão surgir os primeiros dados.
• necessitam de executar outras tarefas enquanto não chegam dados.

Soluções (receção assíncrona):

• *sockets* não bloqueantes ou com *Timeout* – obriga a aplicação a periodicamente realizar uma sequência de tentativas de leitura nos vários *sockets* para verificar se existem dados. Será mais eficiente se for desencadeado apenas quando o sistema operativo alerta o processo para o facto de terem chegado dados (Ex.: sinal SIGIO nos sistemas Unix).

• *threads* ou processos – nesta solução é criado um *thread* ou um processo para ler de cada socket, apenas esse processo ou *thread* fica bloqueado à espera de dados.

• função especifica para monitorizar um conjunto de *sockets* (Ex.: *select*() em C), esta função recebe como argumento um conjunto de *sockets* e desbloqueia quando chegam dados a qualquer um deles (ou se esgota o *Timeout* também fornecido à função).