

Redes de Computadores

Encaminhamento pelo Caminho mais Curto

(1) *Link State Routing*

Departamento de Informática da
FCT/UNL

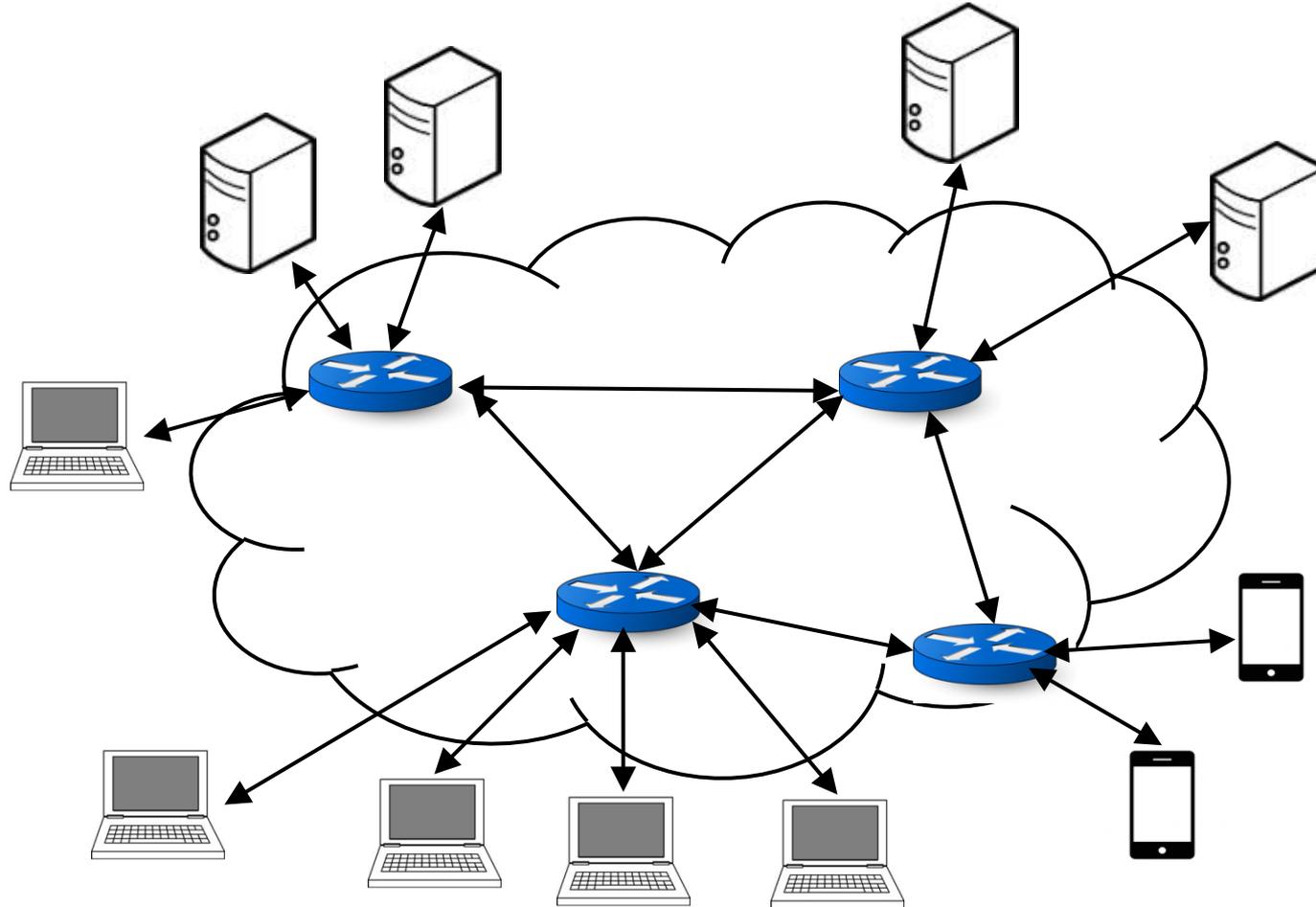
Objetivos do Capítulo

- O encaminhamento numa rede de computadores tem de resolver o problema de determinar o caminho que um pacote deve seguir desde o computador de origem até ao destino
- Numa rede em que existem vários caminhos possíveis existem muitas alternativas
- Neste capítulo vamos conhecer soluções denominadas Encaminhamento pelo Caminho Mais Curto
- Nesta parte do capítulo vamos conhecer uma solução baseada num algoritmo de teoria dos grafos

Make everything as simple as possible, but not simpler.

- Autor: *Albert Einstein (1879-1955)*

O Problema Geral do Encaminhamento



Consiste em escolher um caminho ótimo para os pacotes entre a origem e o destino, que pode ser o mais "direto", mas nem sempre, dependendo da carga da competição pelo mesmo.

Formulação do Problema do Encaminhamento

- Uma rede pode ser modelizada por um grafo, isto é, por um conjunto de nós e um conjunto de arcos, os canais, que interligam os nós
- O problema do encaminhamento é o seguinte: dados quaisquer dois nós X e Y que caminho deve seguir cada pacote com origem em X e destino Y ?
- Se existe mais do que um caminho possível, qual deles escolher?
 - Um grafo em que entre quaisquer dois nós X e Y só existe um caminho único é uma árvore
 - No caso geral o grafo tem a configuração de uma malha e disponibiliza caminhos alternativos

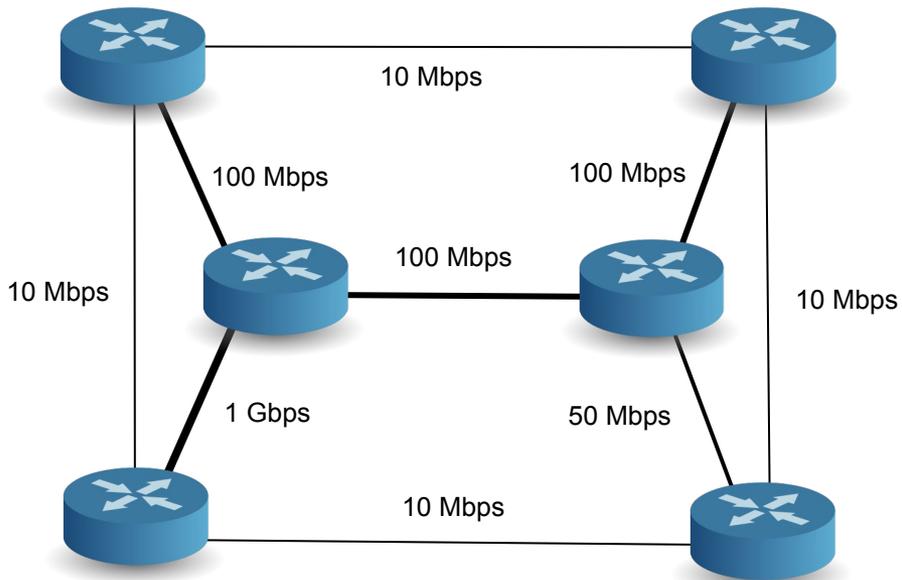
O Problema e uma Solução Possível

- O problema do encaminhamento numa rede de comutação de pacotes é um problema fundamental das redes de computadores que envolve facetas de modelização e otimização, algorítmicas, de engenharia para sua concretização, operacionais, e de planeamento e gestão.
- Uma primeira solução de compromisso relativamente comum consiste em usar encaminhamento pelo caminho mais curto. Esta solução é aceitável quando os caminhos mais curtos estão bem dimensionados para o tráfego existente.

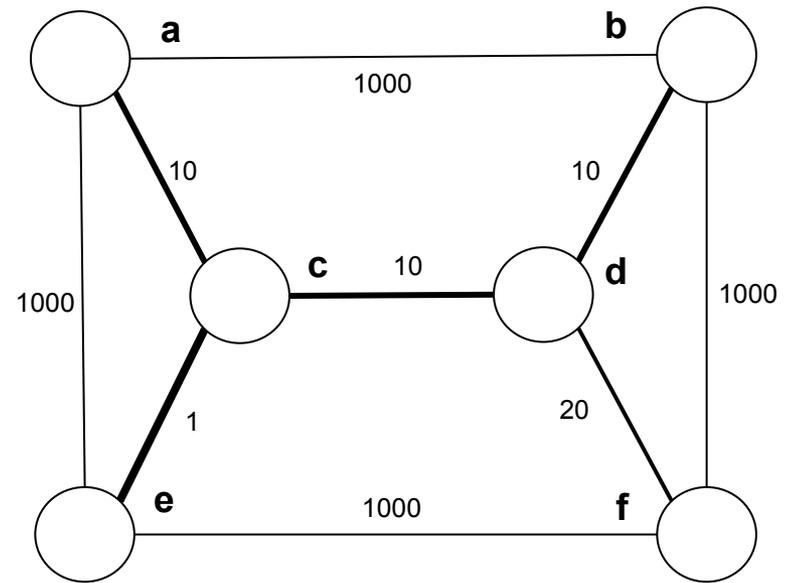
Modelização de Redes com Grafos

- Os nós do grafo são os equipamentos de comutação de pacotes da rede, os nós de comutação, os arcos são os canais
- Custo de um canal é um valor positivo que permite comparar arcos e caminhos
- A função que a cada canal associa um custo é designada por métrica do encaminhamento
- Os canais dizem-se simétricos se o custo é independente da direção do tráfego, o que é uma situação comum

Modelização de Redes com Grafos



(a) - Rede



(b) - Rede com os custos dos canais

SPR - *Shortest Path Routing*

- Problema clássico de otimização em teoria dos grafos
- Algoritmo clássico - proposto por Dijkstra
- Dado um nó origem determina o caminho mais curto para todos os outros nós
- A noção de comprimento de um canal é diferente da noção de custo de um canal, no entanto, vamos assumir que são equivalentes, admitindo que o custo de um arco está de alguma forma relacionado com o seu comprimento

Dados $G = (N, E, cost)$ e $orig$

G é o grafo que modeliza a rede

N é o conjunto dos seus nós

$orig$ o nó inicial, nó a partir do qual se querem conhecer caminhos mais curtos para todos os outros nós

E é o conjunto dos arcos

$cost(x, y)$ é custo do arco que liga diretamente os nós x, y . O custo é sempre positivo.

$cost(x, y) = \infty$ se não existe um arco entre x e y , senão é igual ao custo desse arco

Variáveis Auxiliares

Estavel ou S de stable é o conjunto de nós para os quais já se conhece um caminho mais curto com origem em **orig**

Inicialmente $S = \{\text{orig}\}$

No fim $S = N$

Tentativa ou T de tentative é o conjunto de nós para os quais já se conhece um caminho com origem em **orig** se este existe

Inicialmente $T =$ nós diretamente ligados a **orig**

No fim $T =$ vazio e todos os nós sem caminho para **orig** são colocados em N à distância infinito

Vetor distance [v] - Σ dos custos dos arcos do caminho de **orig** a v

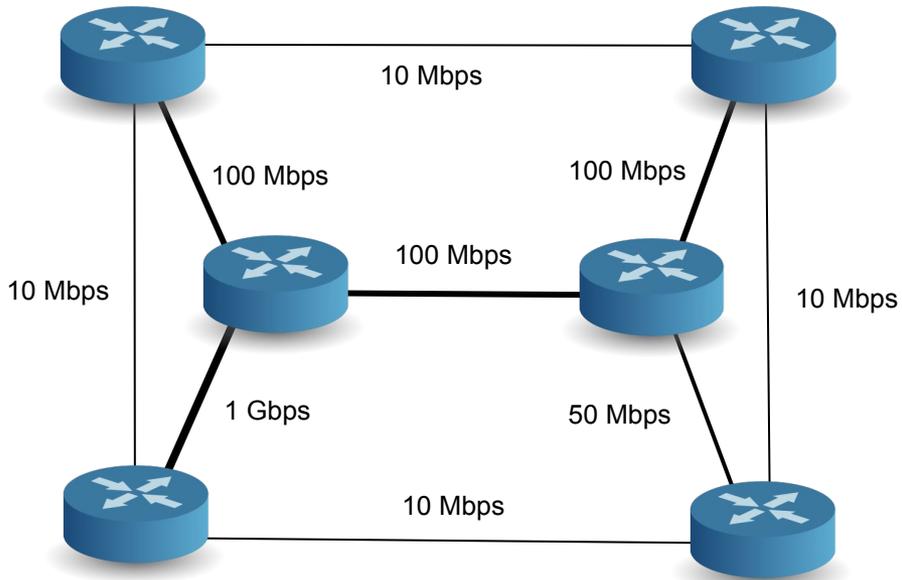
Inicialmente, $\text{distance}[v] = \text{cost}(\text{orig}, v)$ se existe um arco que liga **orig** a v , ou ∞ se não existe

Vetor predecessor [v] - nó que precede v no caminho escolhido de **orig** para v

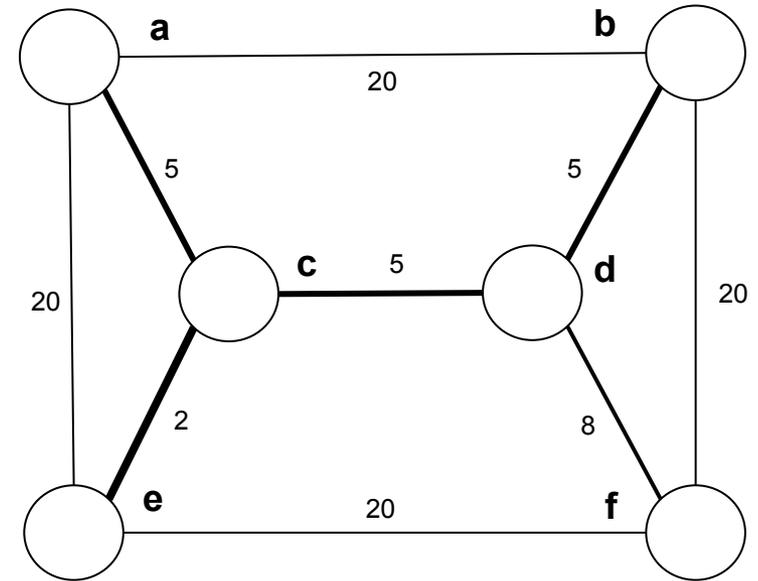
Algoritmo de Dijkstra

```
while ( S ≠ N) {  
    Encontrar p (de pivot) não pertencente a S tal que  
        distance(p) é um mínimo (p está em  
        Tentativa)  
    Juntar p a S // pois não existe nenhum caminho mais curto para w  
    Actualizar distance(v) para todos os nós v  
    adjacentes a p e que ainda não pertencem  
    a S usando:  
        distance(v) = min( distance(v),  
        distance(p) + cost(p,v) )  
    // O novo custo para v é o antigo distance(v) ou o novo  
    // via p (o pivot) caso este seja inferior; não existe nenhum  
    // outro caminho para v mais curto visto que não existe  
    // nenhum outro caminho mais curto para p  
}
```

Dados



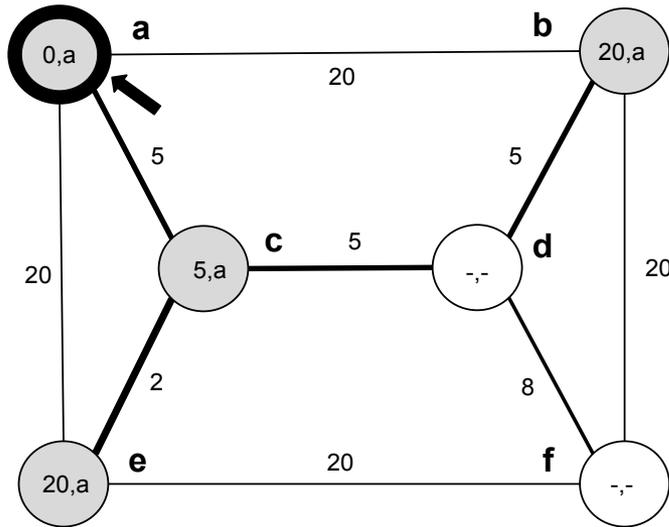
(a) - Rede



(b) - Modelo

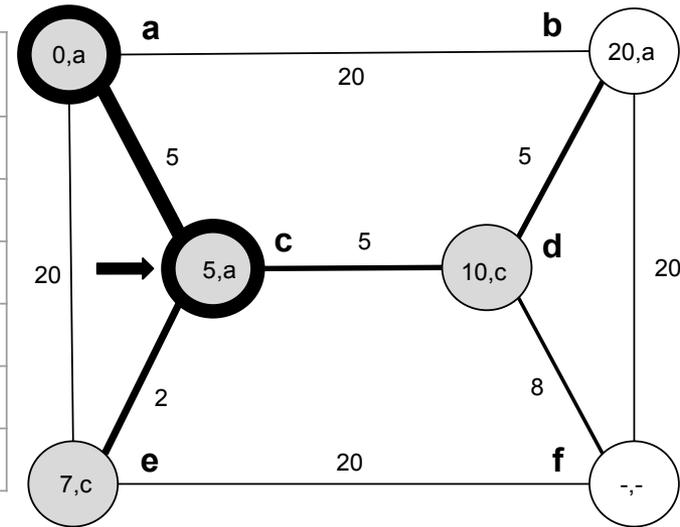
À direita figura o grafo $G = (N, E, cost)$ a orig é o nó a

Inicialização e Primeira Iteração



Estável	Tentativa
(a,0,-)	(c,5,a)
	(e,20,a)
	(b,20,a)

(a) - Estado após a inicialização

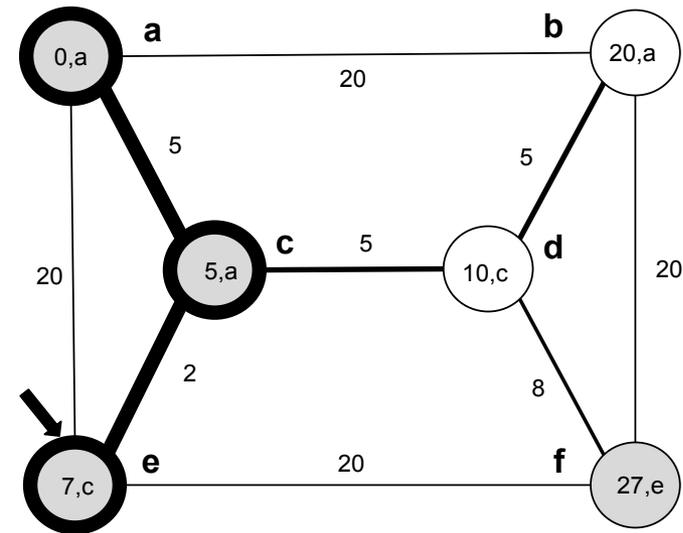


Estável	Tentativa
(a,0,-)	(e,7,c)
(c,5,c)	(d,10,c)
	(b,20,a)

(b) - Estado após a primeira iteração

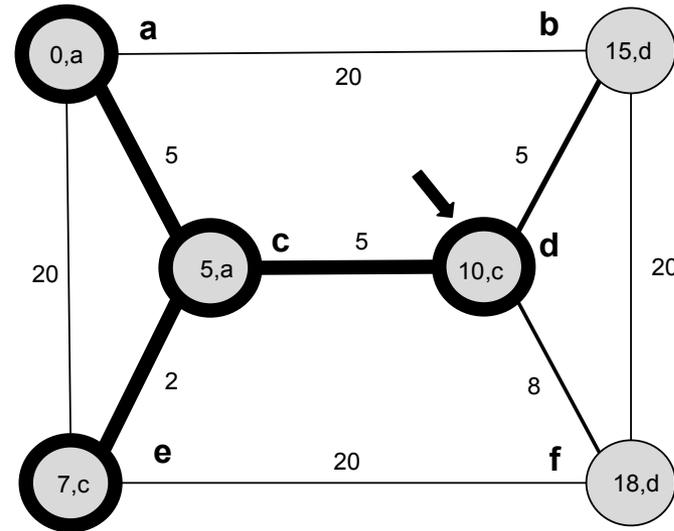
Após a inicialização, o nó c é o primeiro mínimo de Tentativa, o novo pivot, é incluído em Estável, inclui-se em Tentativa o novo nó d diretamente ligado ao pivot e atualiza-se a distância e o predecessor de e visto que se descobriu um melhor caminho para o mesmo via o pivot. Não há alterações em b visto que não se descobriu nenhum caminho mais curto.

Segunda e Terceira Iterações



(a) - Estado após a segunda iteração

Estável	Tentativa
(a,0,-)	(d,10,c)
(c,5,c)	(b,20,a)
(e,7,c)	(f,27,e)

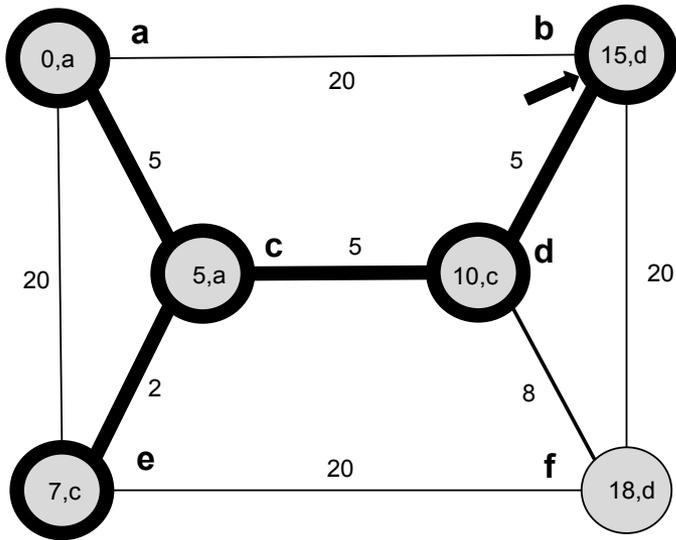


(b) - Estado após a terceira iteração

Estável	Tentativa
(a,0,-)	(b,15,d)
(c,5,c)	(f,18,d)
(e,7,c)	
(d,10,c)	

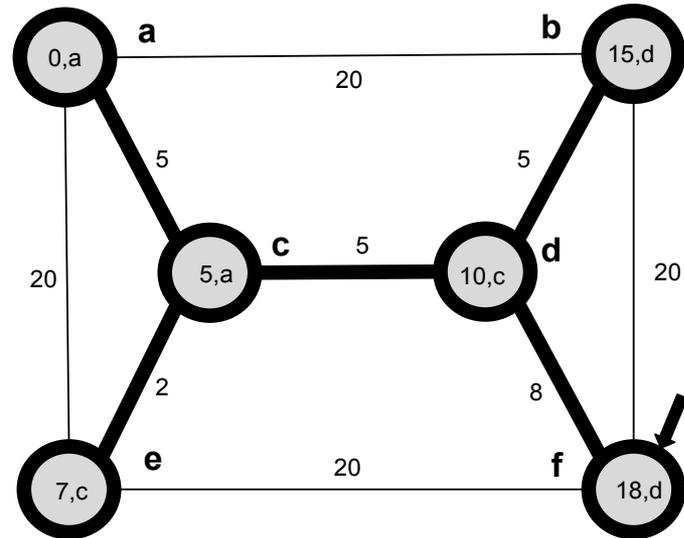
(2) **e** é o novo **pivot**, é incluído em Estável, inclui-se em tentativa o novo nó **f** diretamente ligado ao **pivot**, mas não se descobrem melhores caminhos para os outros nós. (3) **d** é o novo **pivot** que é incluído em Estável, e atualizam-se as distâncias e os caminhos para **b** e **f** via o mesmo visto que são mais curtos.

Quarta e Quinta Iterações



Estável	Tentativa
(a,0,-)	(f,18,d)
(c,5,c)	
(e,7,c)	
(d,10,c)	
(b,15,d)	

(a) - Estado após a quarta iteração



Estável	Tentativa
(a,0,-)	
(c,5,c)	
(e,7,c)	
(d,10,c)	
(b,15,d)	
(f,18,d)	

(b) - Estado após a quinta iteração

(4) **b** é o novo **pivot**, é incluído em **Estável**, mas não se descobrem mais nós a colocar em **Tentativa**, nem caminhos mais curtos para os outros nós. (5) **f** é o novo **pivot** que é incluído em **Estável** e termina-se visto que **Estável** é igual ao conjunto de todos os nós **N**.

No fim conhece-se um árvore de cobertura de **G** de caminhos mais curtos com origem em **a** (**orig**).

Complexidade do Algoritmo

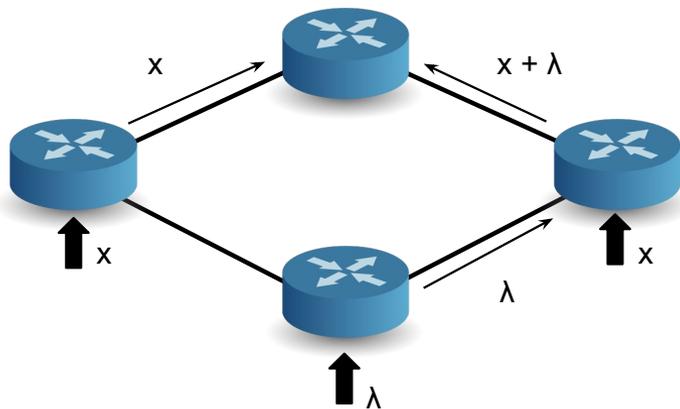
- Em cada uma das n iterações é necessário testar todos os nós que ainda não estão em S
- Do que resulta uma complexidade $O(n^2)$
- Conseguem-se implementações mais otimizadas com custo $O(n \log n)$
- De qualquer forma é um algoritmo pesado e que necessita de informação completa sobre a rede

Que Métricas Usar?

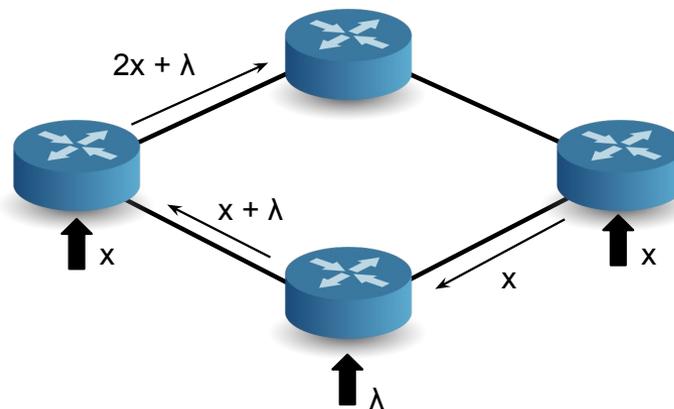
- Para a definição do custo de um canal poder-se-ia usar as suas características essenciais:
 - Débito e tempo de propagação
 - Taxa de erros e dimensão média da fila de espera
- A prática mostrou que se obtém melhor estabilidade na rede usando propriedades que não variam (ver a seguir)
- Por exemplo custo = referência / débito do canal

Exemplo: custo = 1 para 10 Gbps (referência)
= 10 para 1 Gbps,
= 100 para 100 Mbps,
= 1000 para 10 Mbps,
= 10.000 para 1 Mbps

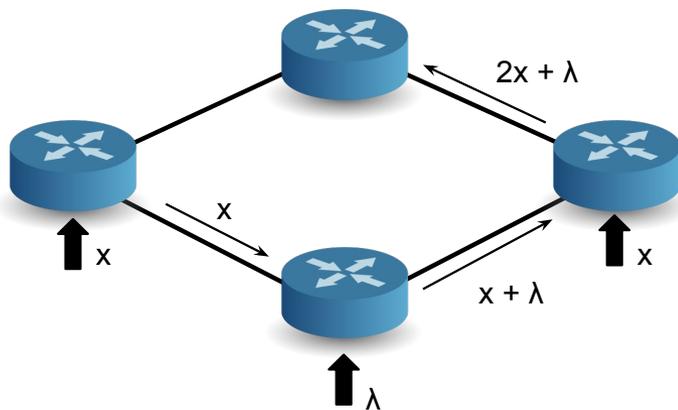
Métricas Variáveis e Instabilidade



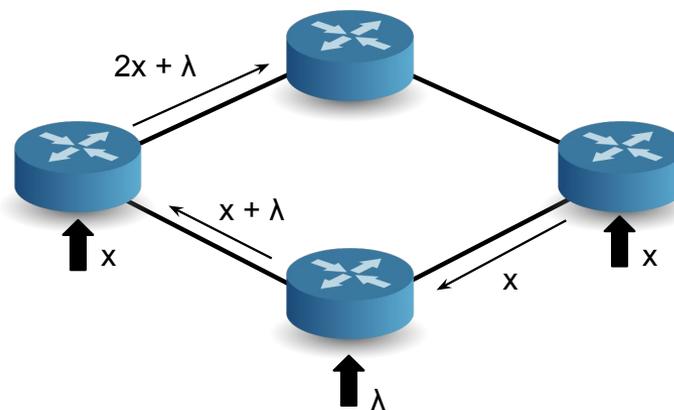
(a)



(b)



(c)

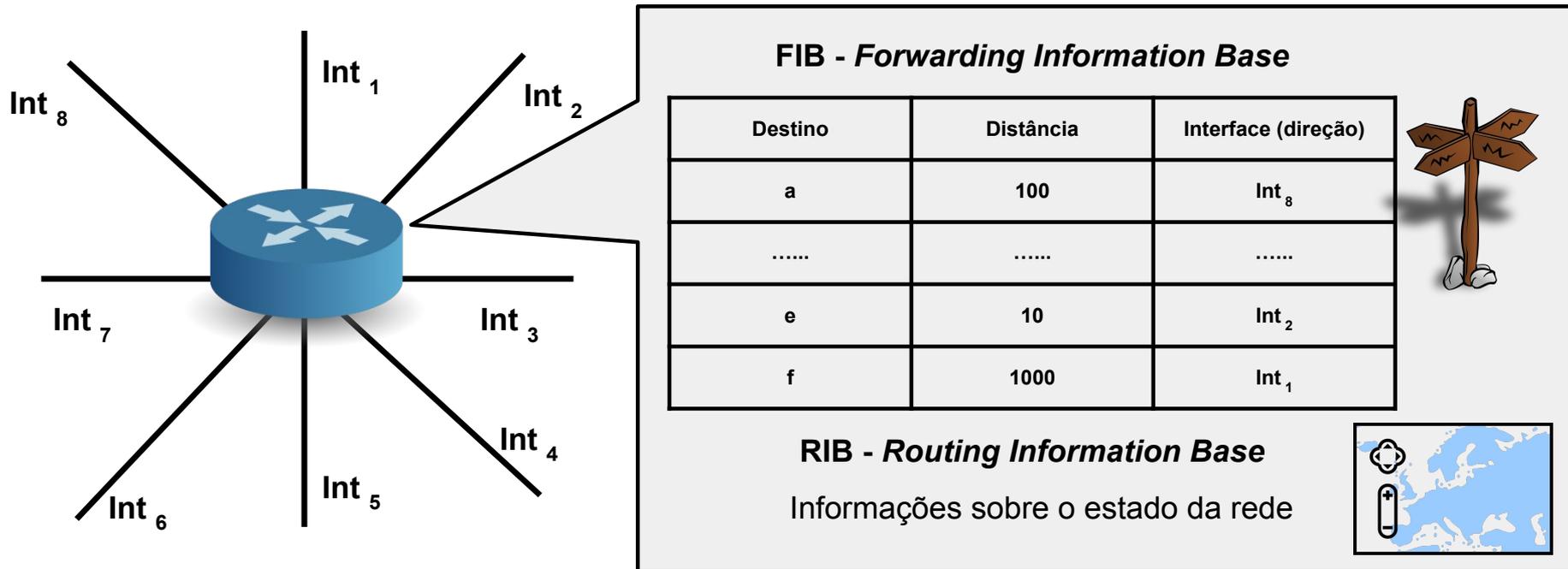


(d)

Conclusões

- Numa rede em malha (onde existem ciclos) é necessário escolher os caminhos seguidos pelos pacotes, pois existe mais do que um caminho para cada destino
- O algoritmo de Dijkstra, um algoritmo clássico de otimização em teoria dos grafos, fornece uma solução que permite determinar um caminho mais curto
 - Baseia-se em estabelecer um modelo da rede como um grafo
 - Determina uma árvore de cobertura de caminhos mais curtos com base na origem
 - Tem uma complexidade aceitável para os computadores modernos
 - Mas exige uma visão completa da rede !

Comutadores Inteligentes ou *Routers*



A FIB é como que um conjunto de tabuletas num cruzamento. Geralmente, a RIB assemelha-se mais a um mapa global da estrada que contém uma visão dos caminhos da origem até ao destino.

Diferença entre RIB e FIB

- Tabela de encaminhamento ou RIB
 - Contém a informação necessária ao funcionamento do protocolo de encaminhamento (*routing*) e permite preencher a FIB
 - Em inglês diz-se *Routing table ou RIB - Routing Information Base*
- Tabela de Comutação ou FIB (*Forwarding Information Base*)
 - Tabela de comutação especial, construída em memória especialmente rápida nos comutadores de alta gama
 - As suas linhas resumem-se a pares (destino, interface, custo)
 - Em inglês diz-se *Forwarding Information Base - FIB*
 - Às vezes a FIB reside diretamente nas placas que recebem os pacotes nos comutadores

Terminologia dos Protocolos de Encaminhamento e dos Computadores

- *Data plane* ou parte dos dados (faceta relacionada com os pacotes de dados)
 - Trata da problemática da comutação de pacotes entre a interface de entrada e a interface de saída
 - Baseia-se numa tabela de comutação
 - Em inglês diz-se *Forwarding Information Base* ou *FIB*
 - Esta tabela é consultada para decidir como transmitir cada pacote
- *Control plane* ou parte do controlo (faceta de controlo)
 - Trata da problemática da informação e protocolos necessários ao controlo do equipamento de comutação
 - Nomeadamente tudo o que lhes permite adquirirem a informação necessária para preencherem a *FIB (Forward Information Base)*
 - Utiliza estruturas de dados dependentes do protocolo e algoritmo usado. Estas designam-se por *RIB (Routing Information Base)*
 - *Com encaminhamento estático, a RIB é preenchida à mão.*

Encaminhamento com Base no Estado dos Canais

- Admitindo que cada comutador conhece o grafo da rede, isto é, o seu estado completo, o algoritmo de Dijkstra permitiria tomar decisões sobre o encaminhamento dos pacotes
- A técnica conhecida por *Link State Routing* permite que todos os nós da rede conheçam a configuração completa da mesma e possam estar de acordo na escolha dos caminhos que os pacotes devem seguir
- Baseia-se na difusão de anúncios de estado (LSA *Link State Announcements*) entre os comutadores

Link-State Routing

- O algoritmo de Dijkstra pressupõe que cada nó conhece o grafo da rede
- Como à partida só é fácil cada nó conhecer a sua vizinhança — que canais e que vizinhos tem — é necessário encontrar uma forma de cada comutador ficar a conhecer a visão completa da rede
- *Link State Routing* baseia-se na ideia de que se cada nó difundir de forma fiável para todos os outros a sua visão sobre a vizinhança, então todos ficarão com uma visão (consistente e correta) da rede

Como Funciona um Comutador LS



Base de dados de LSA, tabela de vizinhos e tabela de encaminhamento (RIB - routing information base)

Comutador originador	Seq.	TTL (horas)	Tipo e valor do LSA
Comut. A	3	9	...
Comut. A	15	9	...
Comut. B	56	10	...
Comut. C	2	10	...
...
Comut. D	19	9	

Tabela de comutação (FIB - Forwarding Information Base)

Destino	Custo	Canal
A	100	C1
B	50	C2
C	1000	C3
D	10	C2
....
M	200	C4

RIB de um Computador LS



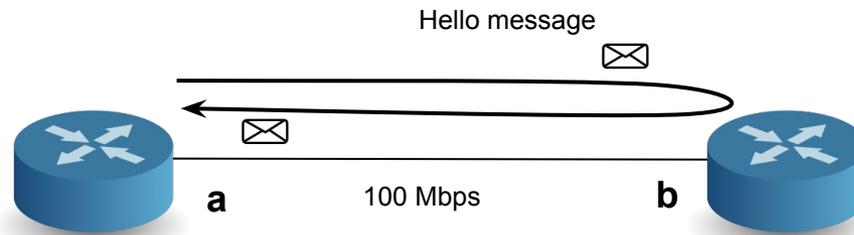
Computador originador do LSA	Sequência	TTL (horas)	Tipo e valor do LSA
Computador a	3	9	...
Computador a	15	9	...
Computador b	56	10	...
...
Computador f	19	9	

Neste caso a RIB chama-se Link State Database ou Adjacencies Database e contém os anúncios de adjacência de todos os computadores. Quando a rede está estável, esta tabela é igual em todos os computadores da rede usando replicação fiel dos anúncios de adjacência ou LSAs (Link State Announcements).

Link-State Routing

- Cada nó (*router*) sabe quais são os seus canais e o respetivo estado
 - O canal está *up* ou *down*?
 - O custo do canal
- Cada nó (*router*) executa um *broadcast*, através de inundação fiável, do estado do canal (*link state reliable broadcast*)
 - Para que cada nó fique com uma visão completa da rede
- Cada nó executa o algoritmo de Dijkstra
 - Calcula os caminhos óptimos (*shortest paths*)
 - ... e reconstrói a sua tabela de encaminhamento
- Protocolos concretos que usam este método
 - Open Shortest Path First (OSPF)
 - Intermediate System - Intermediate System (IS-IS)

Protocolo Hello



- **Sondas periódicas (*beaconing*)**
 - Envio periódico de mensagens "hello"
 - Detecção de falhas após um certo número de ausência de resposta
- **Alguns canais detetam avarias e lançam alarmes**
 - Ausência de portadora por exemplo

Difusão fiável

- Com *Link-State Routing* inicialmente cada nó difunde os canais que conhece e o seu estado
 - Os canais e o seu estado estão na *Link-State Database*
- Sempre que o estado de um canal se altera, é necessário que o nó responsável pelo canal
 - Difunda a alteração a todos os outros nós
 - Como garantir que as notícias chegam a todos ?
- Difusão fiável (*reliable flooding*)
 - Cada nó numera sequencialmente cada mensagem que envia
 - As mensagens são difundidas por inundação (*flooding*)
 - Um nó passa a todos os vizinhos a mensagem que recebe exceto ao vizinho de que a recebeu
 - O número de sequência e a identificação do originador são usadas para detetar os duplicados e as faltas

LSA - Link State Announcement

- Cada comutador emite LSAs
 - No início a dizer quais são os canais que conhece (e.g. canal de a para b operacional e com custo 100)
 - Mas também a dizer os destinos que serve (e.g. os computadores a ele ligados: c1, c2, c3, ...)
 - A seguir sempre que há uma alteração (e.g. canal de a para a down ou com custo ∞) a alteração é difundida
- Cada comutador tem um identificador único
 - Todos os LSAs têm o identificador do emissor original (o originador ou "dono" do LSA)
 - Todos os LSAs têm um número de sequência por ordem crescente colocado pelo seu originador

Link State Database

- O recetor de um LSA
 - Fica a saber que o originador existe (que até pode não ser seu vizinho direto)
 - Fica a saber se já conhece ou não esse LSA pelo seu número de sequência
 - Só lhe interessa guardar a última versão
 - O conjunto dos últimos LSAs recebidos ficam na LSA database
- A LSA database de cada comutador contém a sua visão da rede
 - Funciona como um sistema de emissão de notícias
 - A base de dados de notícias representa o último estado do sistema conhecido
 - A base de dados está replicada em todos os comutadores

Fiabilidade da Difusão

- *Reliable flooding*

- Assegura que todos os nós recebem as notícias
- ... na sua última versão

- **Desafios**

- *Packet loss*
- Chegadas fora de ordem

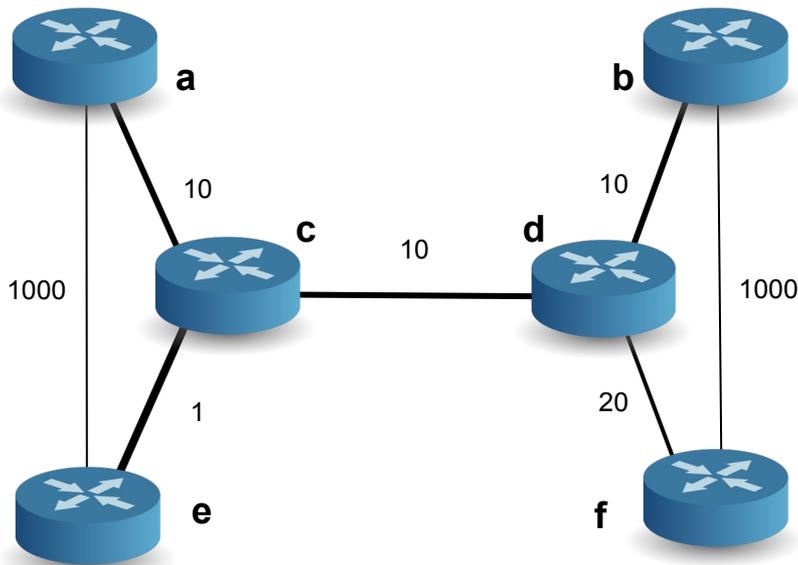
- **Soluções**

- *Acknowledgments* e retransmissões
- Números de sequência
- Time-to-live (TTL) em cada pacote
- Se houver inconsistência dos números de sequência, os nós sincronizam-se para a versão comum mais avançada

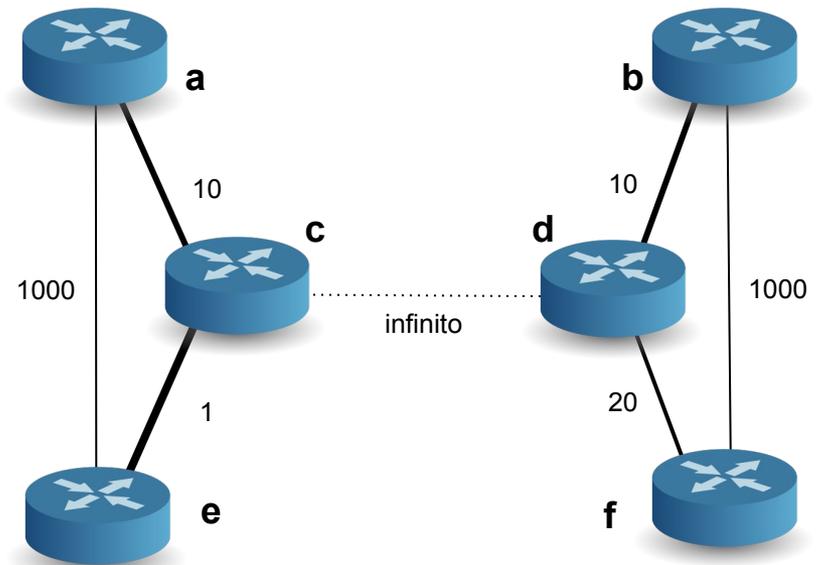
Algoritmo de Difusão Fiável de LSA

- $LSA(Orig, s, notícia)$
 - $Orig$ é o originador do LSA
 - s é o número de sequência
 - $notícia$ é o valor do LSA
- A envia $LSA(Orig, s, notícia)$ ao seu vizinho B
 - B verifica se conhece $Orig$ e se não conhece, executa `database.put(Orig, s, notícia)`, envia `ACK` a A e faz flooding do LSA
 - Se conhece vai buscar o número de sequência do último LSA de $Orig$ que conhece (seja i esse número)
 - Se $i = s$, o LSA é um duplicado, envia `ACK` a A
 - Se $i > s$, A está atrasado e B envia-lhe o seu último LSA conhecido sobre $Orig$
 - Se $i < s$, B está receber notícias novas, `database.put(Orig, s, notícia)`, envia `ACK` a A e faz flooding do LSA

Disseminação Fiável e Partições



(a) - Rede sem partições



(b) - Rede particionada em duas partições

Quando se dá uma partição, os comutadores de cada lado, logo que recebem o anúncio de que o canal que liga c a d foi abaixo, passam a considerar os comutadores da outra partição à distância infinita. Ambas as partições passam a evoluir independentemente. Quando o canal volta a estar operacional, é necessário ressincronizar as duas partições imediatamente.

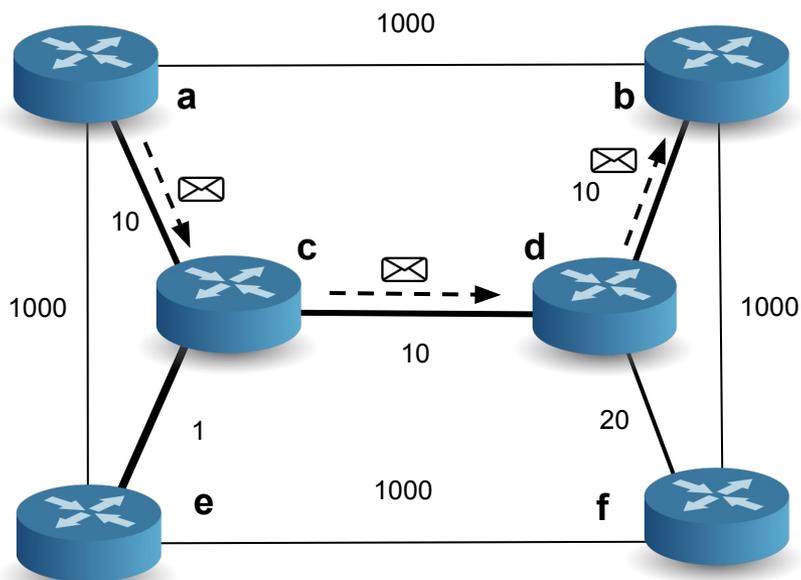
Alterações do Estado da Rede

- Na sequência de uma alteração de estado, os nós ficam com visões distintas do estado da rede
 - Têm tabelas de encaminhamento inconsistentes e os pacotes podem entrar em ciclo
 - O TTL dos pacotes garante que estes não ficam eternamente no interior da rede
 - Mas durante essa instabilidade perdem-se pacotes e desperdiça-se capacidade da rede
- Diz-se que a rede converge quando todos os nós refletem a mesma visão da rede e têm tabelas de encaminhamento coerentes

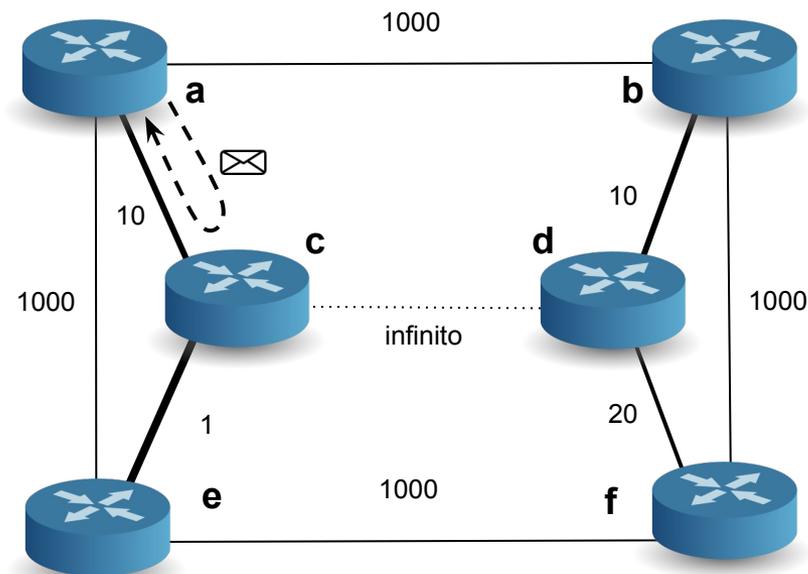
Noção de Convergência da Rede

- O tempo que demora a que todos os nós tenham a mesma visão da rede e tabelas de encaminhamento coerentes
 - Depende do tempo de deteção das alterações
 - do tempo de propagação das mensagens
 - do tempo de execução do algoritmo de Dijkstra
 - e do tempo necessário para alterar as novas tabelas de encaminhamento
 - em nós de grande capacidade e redes realistas é hoje em dia inferior a 1 segundo

Avarias, Ciclos e Convergência



(a) - Rede sem avarias



(b) - Rede após a avaria

Inicialmente a envia pacotes para b via c. Após a avaria do canal c para d, o nó c passa a enviar os pacotes para b via a. Enquanto a não souber da avaria continua a enviar os pacotes para b via c e esses pacotes entram em ciclo, perdem-se e desperdiçam a capacidade do canal de c para a. Quando a rede converge, a vai para b pelo canal direto.

Conclusões

- O algoritmo de Dijkstra permite calcular os caminhos mais curtos a partir de um qualquer nó do grafo da rede
- Mas requer que cada comutador conheça toda a rede
- *Link-State Routing* baseia-se na utilização de difusão fiável para propagar para todos os comutadores a visão completa do estado da rede
- É um algoritmo distribuído que introduz inconsistências durante a convergência, mas esta pode ser relativamente rápida