Elementy směrování jako grafy a jejich aplikace

Petr Holub, hopet@ics.muni.cz

Přepínání a směrování

- Přepínání
  - L2 vrstva – linková (Ethernet), cut through
  - backward-learning
- Směrování
  - L3 vrstva – síťová (IP), store and forward
  - distance vector protokoly:
    - RIPv1 a v2, IGRP
    - BGP do jisté míry
  - link state protokoly
    - OSPF

Hierarchické směrování

- Dělení adres:
  oblasti (region) - obvody (district) - procesy
- rgn[x] definuje souseda, přes nějž se dostanu do regionu *x*
  dstr[y] definuje souseda, přes nějž se dostanu do obvodu *y*
  prs[z] definuje souseda, přes nějž se dostanu do procesu *z*
- up[k] definuje, jestli je soused *k* naživu

Hierarchické směrování (2)

```
process p [i: 0..m-1, {i is the process region}
           j: 0..n-1, {j is the process district}
           k: 0..r-1  {k is the process}
          ]

inp N    : set {[i',j',k'] | p[i',j',k'] is a neighbor of p[i,j,k]},
    up   : array [N] of boolean,
    rgn  : array [0..m-1] of N,
    dstr : array [0..n-1] of N,
    prs  : array [0..r-1] of N
var x : 0..m-1,
    y : 0..n-1,
    z : 0..r-1
par q : N

begin
        true -> {generate a data(x, y, z) msg and route it}
                x, y, z := any, any, any;
                RTMSG

□       rcv data(x,y,z) from p[g] ->
                {route the received data(x, y, z) msg}
                RTMSG
end
```

RTMSG

```
if x/=i ∧                  up[rgn[x]] ->
                                  send data(x, y, z) to p[rgn[x]]
□  x/=i ∧                  ~up[rgn[x]] ->
                                  {nonreachable dest.} skip
□  x=i ∧ y/=j ∧            up[dstr[y]] ->
                                  send data(x, y, z) to p[dstr[y]]
□  x=i ∧ y/=j ∧            ~up[dstr[y]] ->
                                  {nonreachable dest.} skip
□  x=i ∧ y=j ∧ z/=k ∧      up[prs[z]] ->
                                  send data(x, y, z) to p[prs[z]]
□  x=i ∧ y=j ∧ z/=k ∧      ~up[prs[z]] ->
                                  {nonreachable dest.} skip
□  x=i ∧ y=j ∧ z=k                   ->
                                  {arrived at dest.} skip
fi
```

Použití implicitní brány

```
if gtwy = k  ->  RTMSG
□  gtwy /= k  ->
        if (x/=i ∨ y/=j) ∧ up[prs[gtwy]]  ->
                send data(x, y, z) to p[prs[gtwy]]
        □  (x/=i ∨ y/=j) ∧ ~up[prs[gtwy]]  ->
                {nonreachable} skip
        □  (x=i ∧ y=j) ∧ z=k  ->  {arrived} skip
        □  (x=i ∧ y=j) ∧ z/=k ∧ up[prs[z]]  ->
                send data(x, y, z) to p[prs[z]]
        □  (x=i ∧ y=j) ∧ z/=k ∧ ~up[prs[z]]  ->
                {nonreachable} skip
        fi
fi
```

Náhodné směrování

- Příklad se směrovací tabulkou, kde pro každý cíl máme 2 možné uzly, přes něž se bude posílat
  rtb[d, 0] = nejaky_soused1
  rtb[d, 1] = nejaky_soused2

Náhodné směrování (2)

```
process p [i: 0..n-1]

const hmax

inp N    : set { g | p[g] is a neighbor of p[i] },
    up   : array [N] of boolean,
    rtb  : array [0..n-1, 0..1] of N
var x : 0..1,          {random choice}
    d : 0..n-1,        {ultimate destination}
    h : 0..hmax        {# hops remaining = TTL}
par q : N

begin
        true                              -> d, h := any, any;
                                             RTMSG

□       rcv data(d, h) from p[g]          -> RTMSG
end
```

RTMSG

```
if d=i                                -> {arrived at dest.} skip
□  d/=i ∧ h=0                         -> {nonreachable dest.} skip
□  d/=i ∧ h>0 ∧ (d in N ∧ up[d])      ->
                                      send data(d, h-1) to p[d]
□  d/=i ∧ h=1 ∧ ~(d in N ∧ up[d])     ->
                                      {nonreachable dest.} skip
□  d/=i ∧ h>1 ∧ ~(d in N ∧ up[d])     ->
       x := random;
       if up[rtb[d, x]]               ->
                          send data(d, h-1) to p[rtb[d, x]]
        □  ~up[rtb[d, x]] ∧ up[rtb[d, 1-x]]  ->
                          send data(d, h-1) to p[rtb[d, 1-x]]
        □  ~up[rtb[d, x]] ∧ ~up[rtb[d, 1-x]]  ->
                          {nonreachable dest.} skip
       fi
fi
```

Distribuované směrování

- Skutečně distribuované v síti
  - nikdo se nesnaží znát topologii celé sítě
- Každý se snaží znát pro daný cíl pouze souseda, přes něhož se dostane k cíli nejkratší cestou
  - ohodnocení hran mohou být
    - počty skoků (hop count) – RIP
    - jiné parametry (šíka pásma, zpoždění, zátěž, spolehlivst, MTU) – IGRP
  - svoji znalost si každý směrovač odvodí pouze ze znalostí přilehlých směrovačů

Distribuované směrování

- rtb[d] nejlepší soused pro směrování zprávy do p[d]
  cost[d] počet skoků při poslání zprávy přes p[rtb[d]]
- počet procesů v síti je *n*, číslováno 0..*n*-1 => nekonečno můžeme definovat jako *n*
- demonstrujeme na síti s všemi hranami ohodnocenými 1

Distribuované směrování (2)

```
process p [i: 0..n-1]

inp N     : set { g | p[g] is a neighbor of p[i] },
    up    : array [N] of boolean
var rtb         : array [0..n-1] of N,
    cost, c     : array [0..n-1] of 0..n,
    d           : 0..n-1,
    f, h        : N,
    finish      : boolean
par q : N

begin
        true                              -> d := any, any;
                                             RTMSG

□       rcv data(d) from p[g]             -> RTMSG
□       true                              -> SNDCOST
□       rcv upd(c) from p[g]              -> UPDRTB

end
```

RTMSG

```
if d=i                                          -> {arrived} skip

□  d/=i ∧ (cost[d]<n ∧ up[rtb[d]])    ->
                        send data(d) to p[rtb[d]]

□  d/=i ∧ ~(cost[d]<n ∧ up[rtb[d]])   ->
                        {nonreachable} skip

fi
```

SNDCOST

- funkce NEXT(N, h) vrací následující prvek z množiny N po prvku h (umí množinou cyklit)
- h je libovolný prvek z N

```
f := NEXT(N, h);

do f/=h  ->
        if up[f]        -> send upd(cost) to p[f]
        □  ~up[f]       -> skip
        fi; f := NEXT(N, f)
od;

if up[h]        -> send upd(cost) to p[h]
□  ~up[f]       -> skip
fi
```

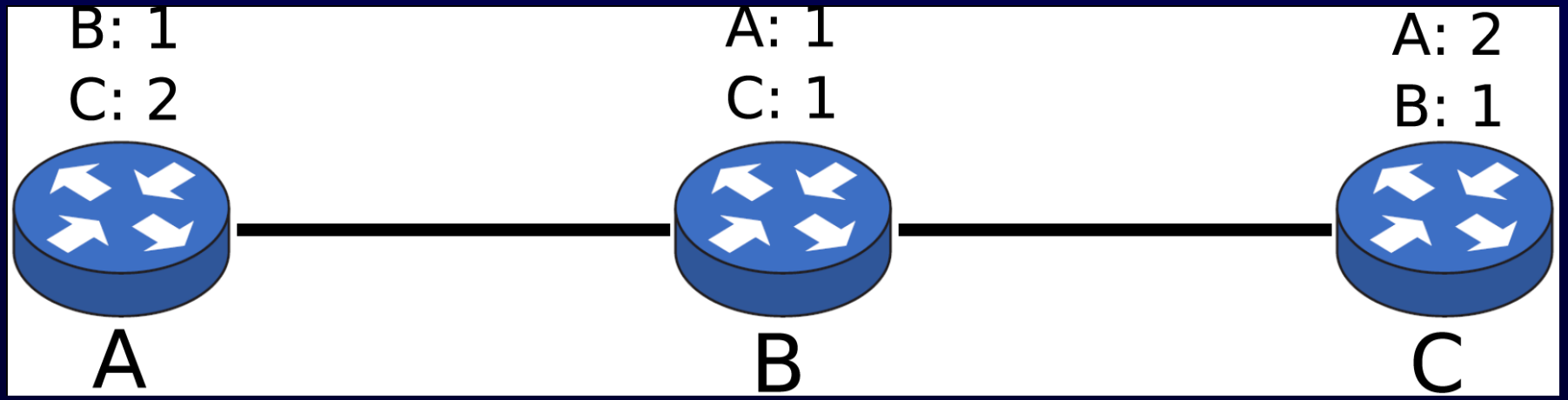
UPDRTB

- Aktualizace rtb[] a cost[]

```
d, finish := 0, false;

do ~finish  ->
        if (d=i)          -> cost[d] := 0
        □  (d/=i) ∧
           (rtb[d]=g ∨ cost[d]>c[d]+1 ∨ ~up[rtb[d]]) ->
               rtb[d], cost[d] := g, min(n, c[d]+1)
        □  (d/=i) ∧
           ~(rtb[d]=g ∨ cost[d]>c[d]+1 ∨ ~up[rtb[d]]) ->
               skip
        fi;

        if d < n-1        -> d := d+1;
        □  d = n-1        -> finish := true
        fi
od
```

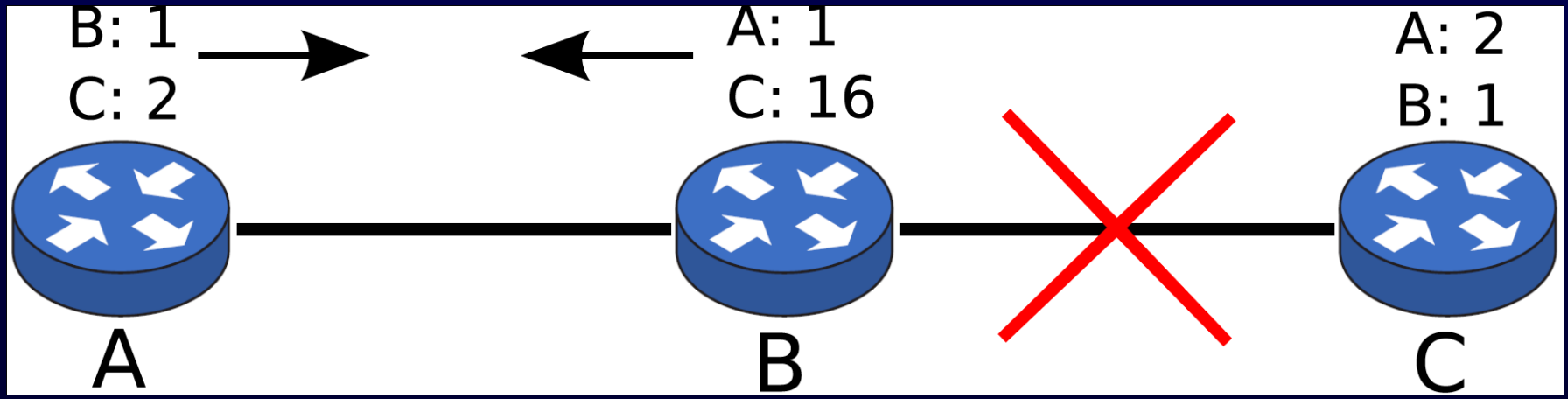
Problém „Count to Infinity“

- Zacyklení distance-vector protokolu
  - Bellman-Ford sám o sobě nezabraňuje smyčkám
  - zabráníme pomocí split-horizon event. s poisoned reverse



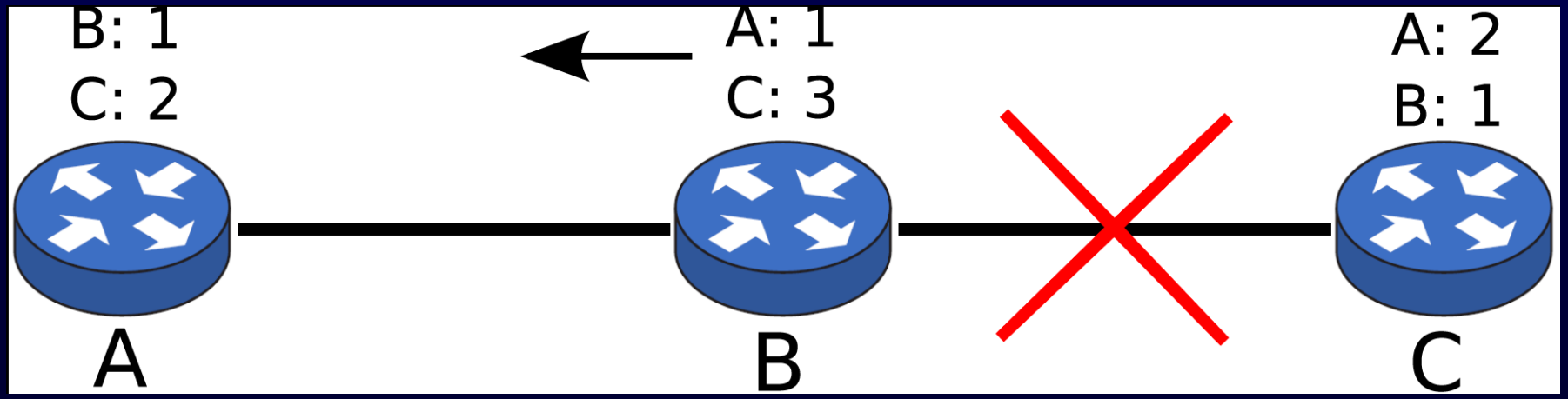
Problém „Count to Infinity“

- Zacyklení distance-vector protokolu
  - zabráníme pomocí split-horizon event. s poisoned reverse



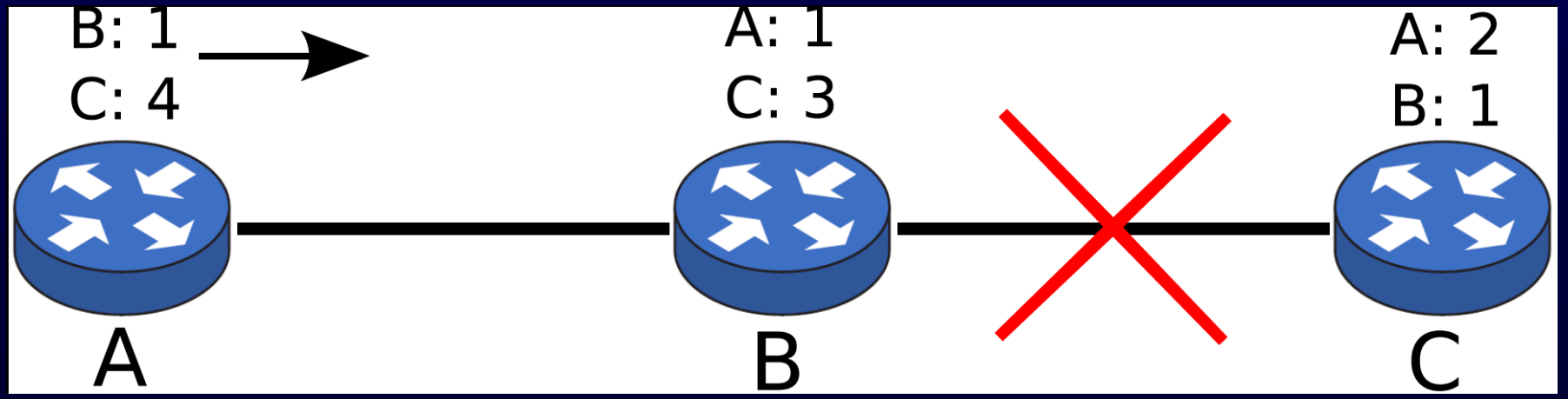
Problém „Count to Infinity“

- Zacyklení distance-vector protokolu
  - zabráníme pomocí split-horizon event. s poisoned reverse



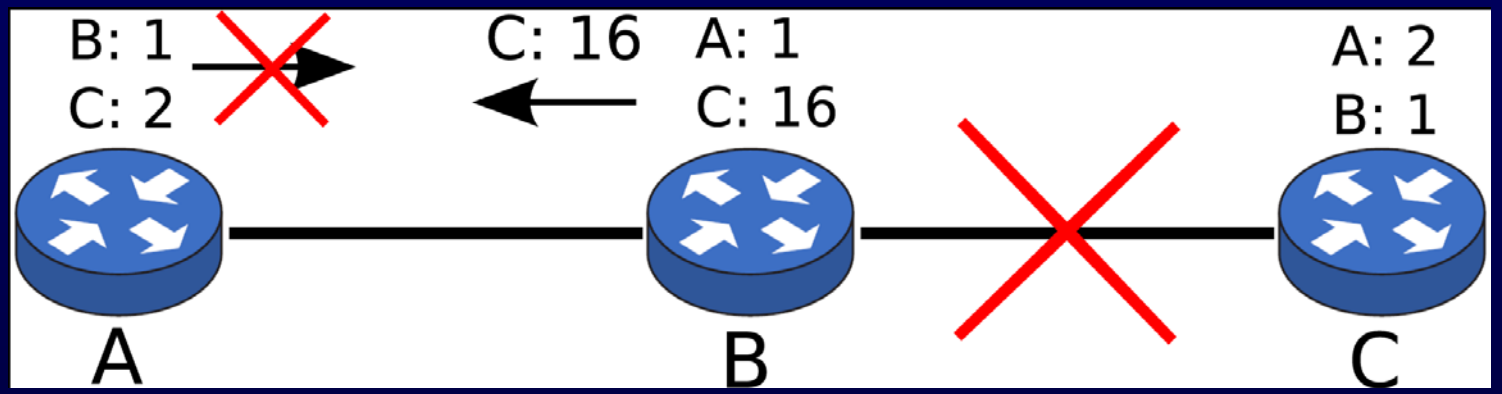
Problém „Count to Infinity“

- Zacyklení distance-vector protokolu
  - zabráníme pomocí split-horizon event. s poisoned reverse

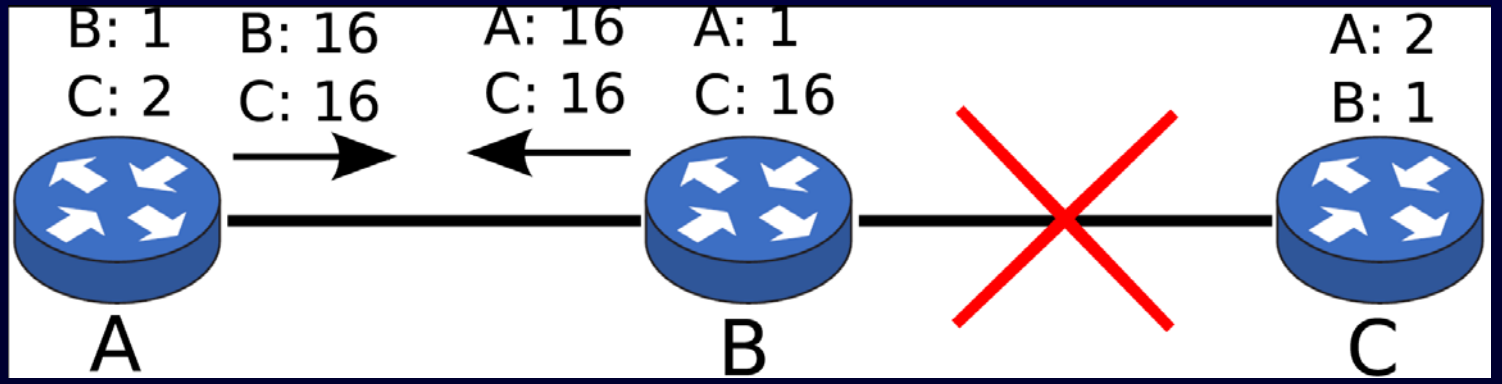


Split horizon

- Neposílá se zpět tomu, od koho jsme se cestu naučili



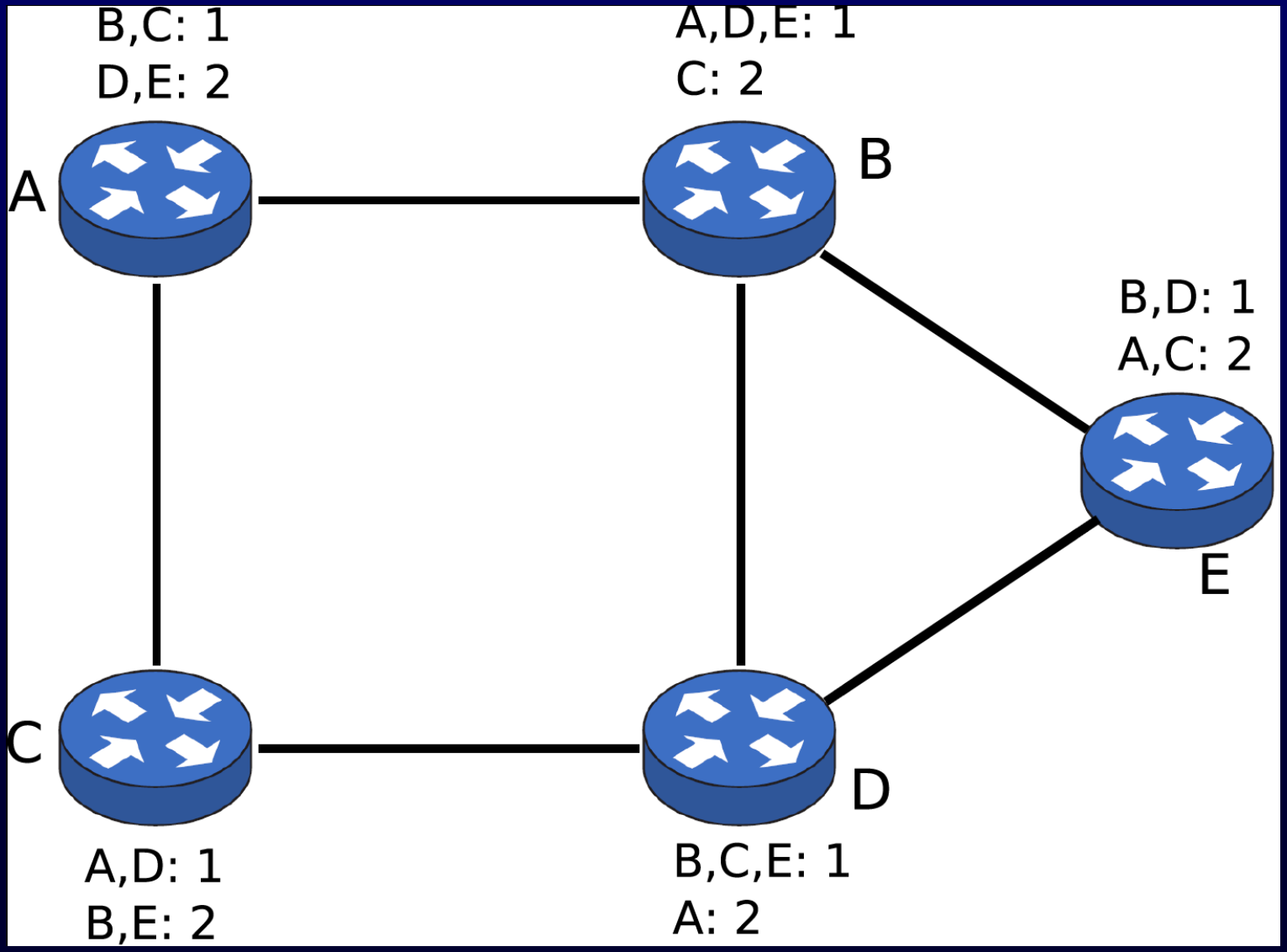
- Poisoned reverse posílá zpět nekonečno



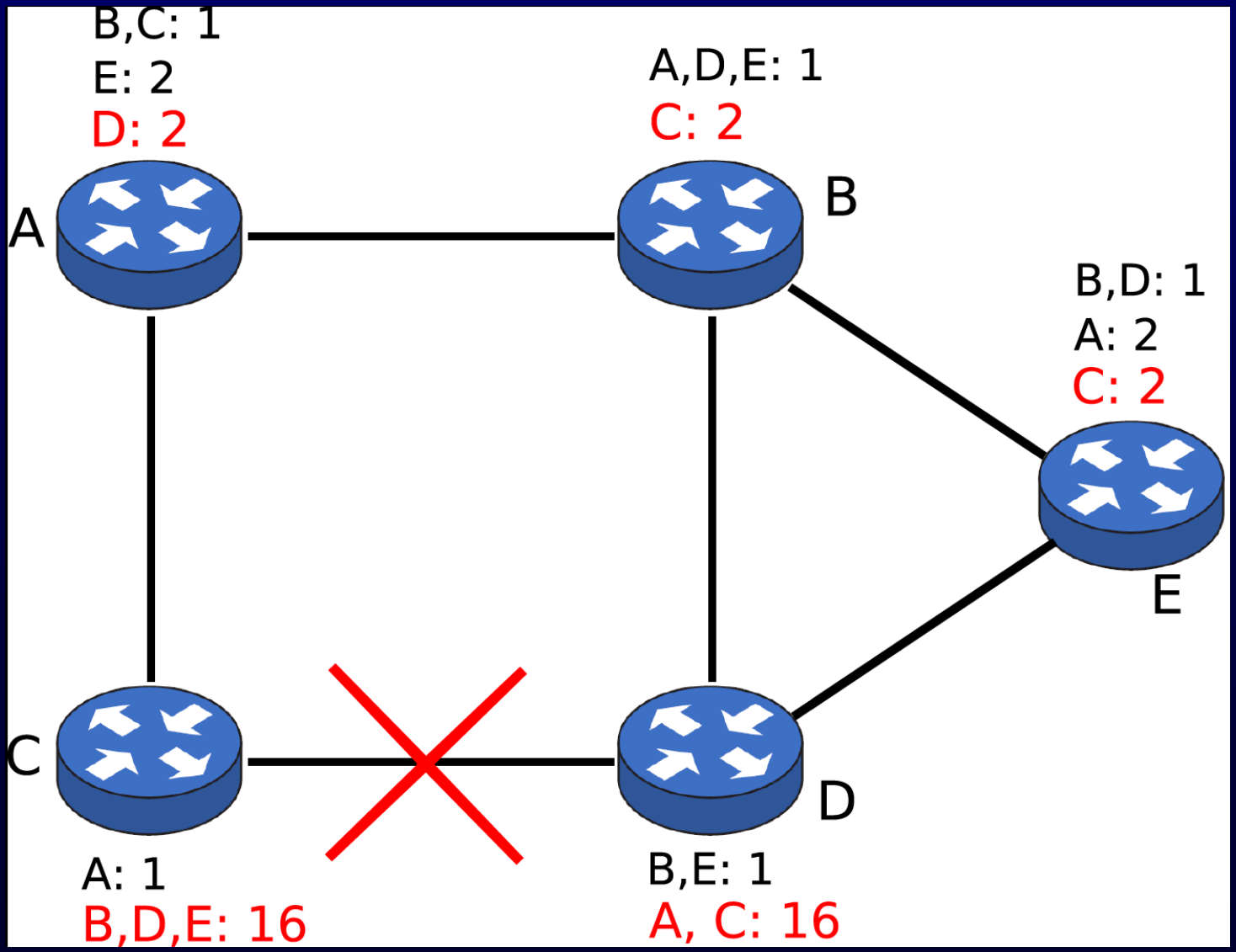
Routing Information Protocol (RIP)

- Implementuje
  - hop count jako metriku linky
  - split horizon
  - holddown
    - počká, než se síť stabilizuje
- RIPv2
  - podpora Classless Inter-Domain Routing (CIDR)
    - posílání informací o prefixech - agregace

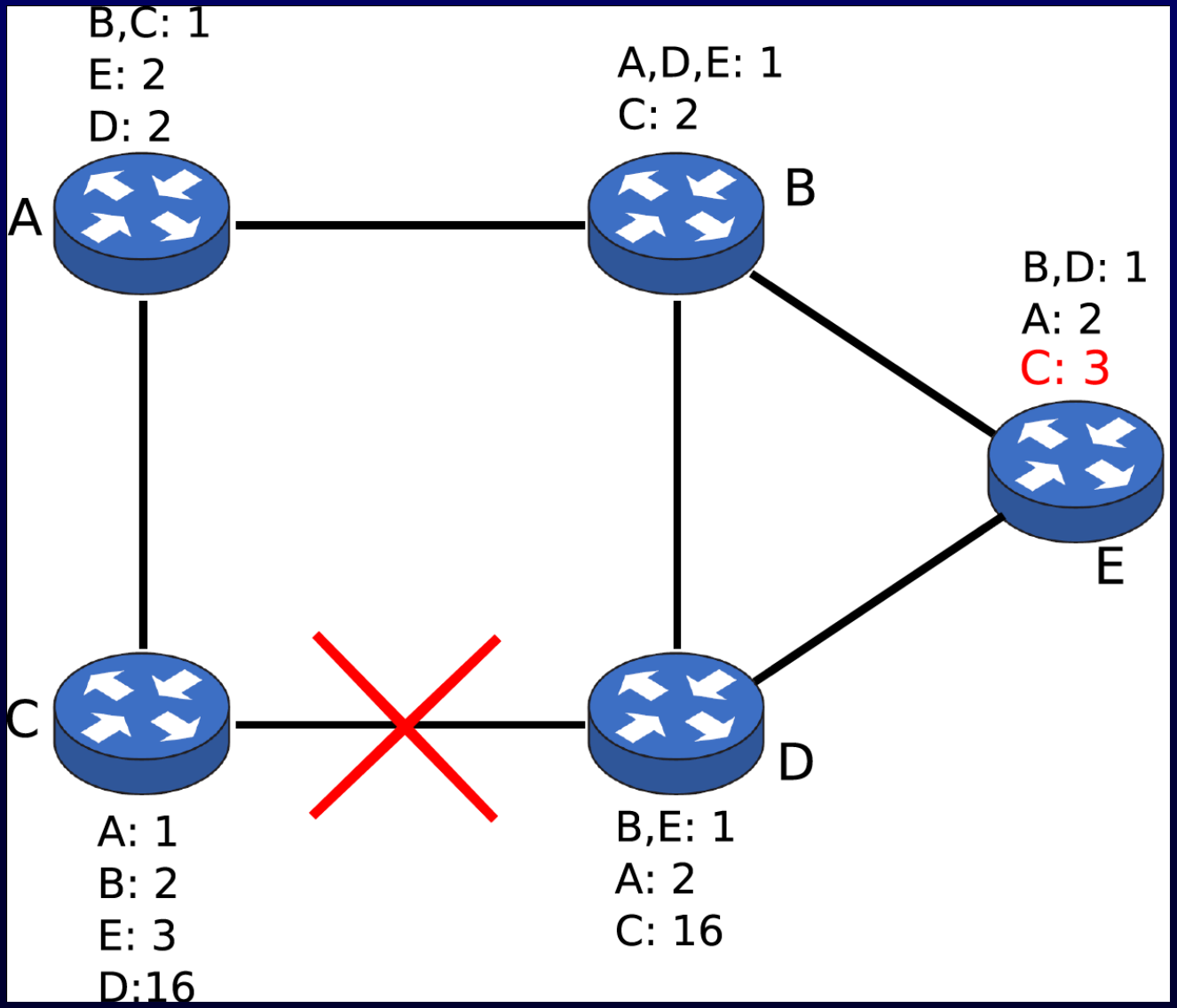
RIP – chování při výpadku



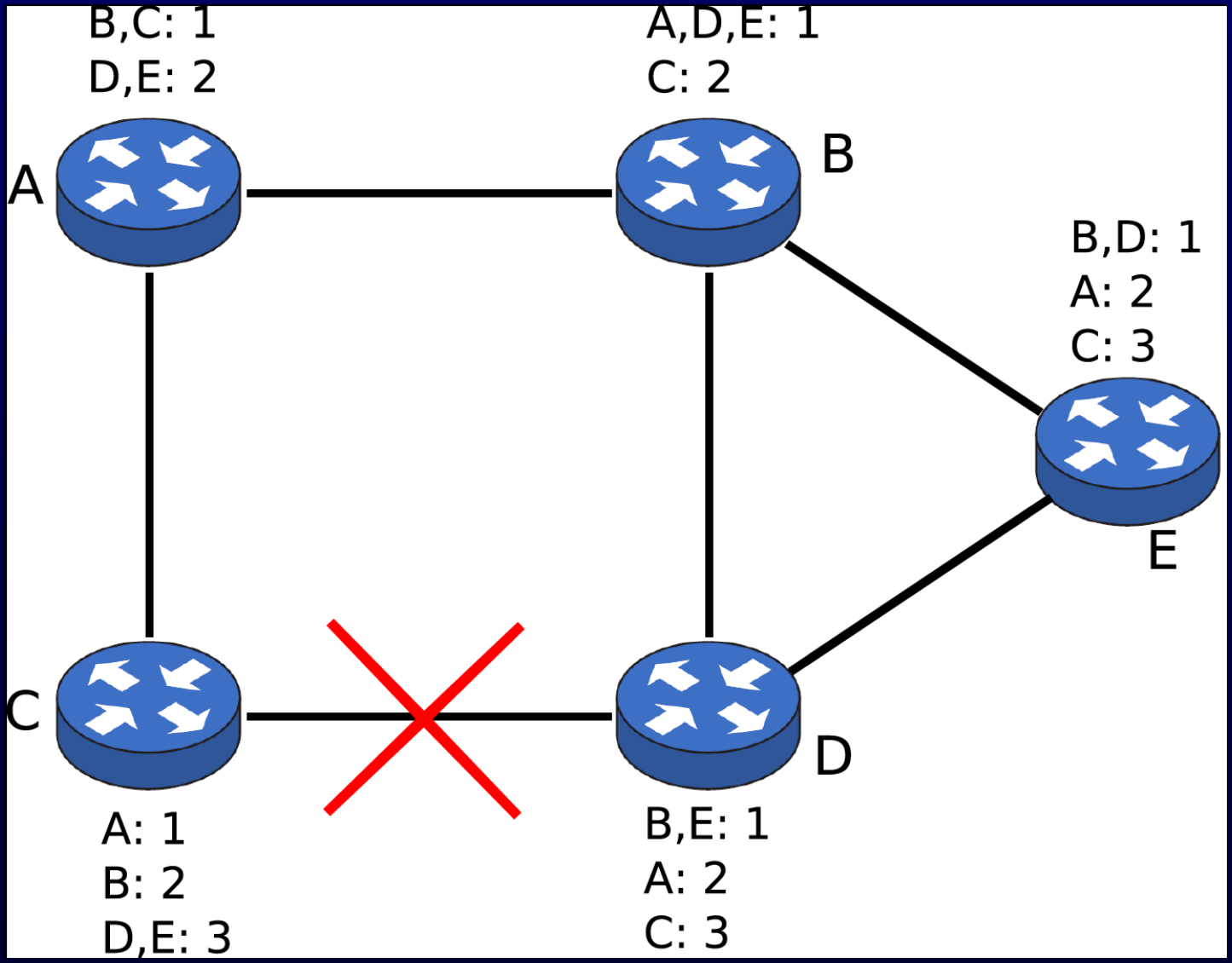
RIP – chování při výpadku



RIP – chování při výpadku



RIP – chování při výpadku



Přepínání

- Přepínače se učí z procházejícího provozu
  - pracují na úrovni MAC adres
  - pokud dostane neznámou adresu, chová se jako hub (rozesílá na všechna rozhraní s výjimkou toho, přes které přijal)
  - problém s cykly v síti

Backward learning

```
process p [i: 0..n-1]

const hmax, vmax

inp N     : set { g | p[g] is a neighbor of p[i] },
    up    : array [N] of boolean
var rtb         : array [0..n-1] of N,
    cost        : array [0..n-1] of 0..n-1,
    valid       : array [0..n-1] of 0..vmax,
    src, dst    : 0..n-1,
    h           : 0..hmax,      {#hops travelled}
    x, y        : N,            {random neighbors}
    flag        : boolean
par q : N

begin
        true              -> src, h, dst := i, 0, any; RTMSG
□       rcv data(src, h, dst) from p[g]       -> RTMSG; UPDRTB
□       true                                  -> UPDRTB'
end
```

RTMSG

```
if dst=i                          -> {arrived} skip
□  dst/=i ∧ h=hmax                -> {nonreachable} skip
□  dst/=i ∧ h<hmax ∧ (dst in N ∧ up[dst])    ->
                        send data(src, h+1, dst) to p[dst]
□  dst/=i ∧ h=hmax-1 ∧ ~(dst in N ∧ up[dst]) ->
                        {nonreachable} skip
□  dst/=i ∧ h<hmax-1 ∧ ~(dst in N ∧ up[dst]) ->
       if up[rtb[dst]]            ->
              send data(src, h+1, dst) to p[rtb[dst]]
       □  ~up[rtb[dst]]           ->
              x := random;
              y := NEXT(N, x);
              do ~up[y] ∧ y/=x  ->  y := NEXT(N, y) od;
              if up[y]           ->
                     send data(src, h+1, dst) to p[y];
                     rtb[dst], cost[dst], valid[dst] := y, n-1, 0
              □  ~up[y]          ->  {nonreachable} skip
              fi
       fi
fi
```

UPDRTB

```
if cost[src]>=h          ->
       rtb[src], cost[src], valid[src] := g, h, vmax

□  cost[src]<h           -> skip

fi
```

UPDRTB'

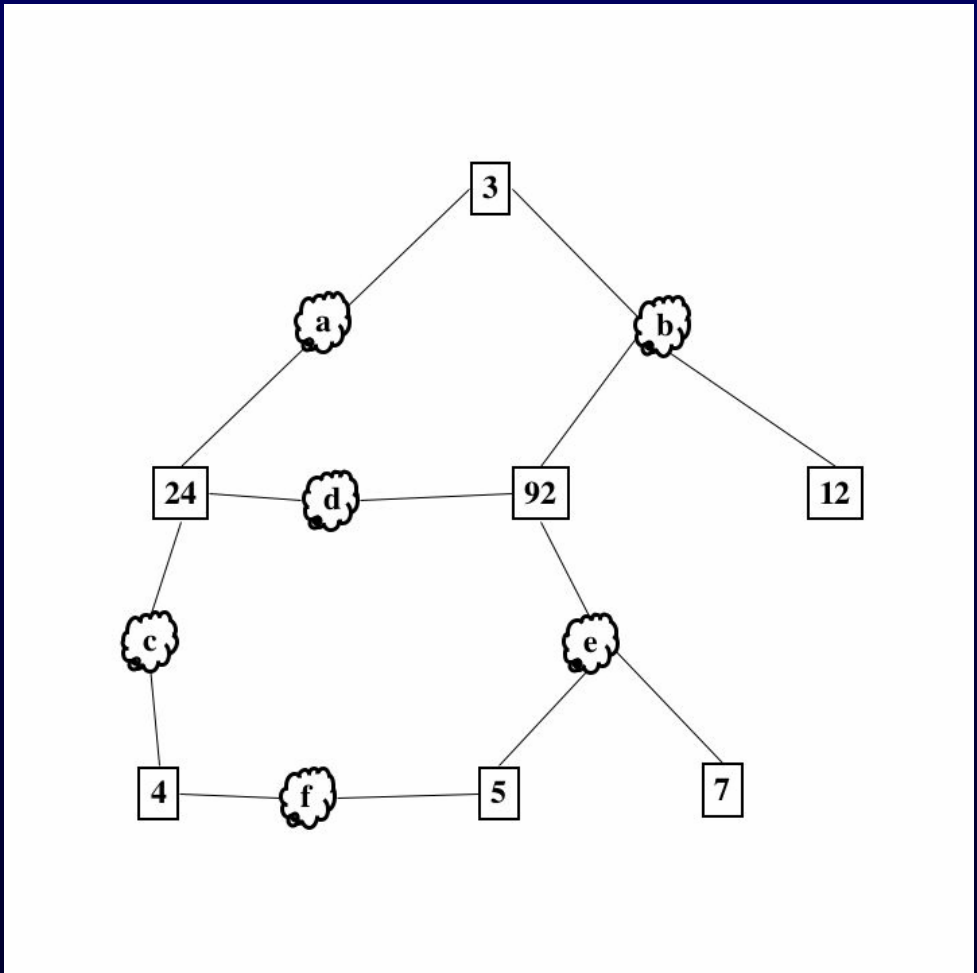
```
flag, dst := true, 0;

do flag  ->
        valid[dst] := max(0, valid[dst]-1);

        if valid[dst]=0         ->  cost[dst] := n-1
        □  valid[dst]/=0        ->  skip
        fi;

        if dst < n-1    ->      dst := dst+1
        □  dst = n-1    ->      flag := false
        fi
od
```

Přepínání

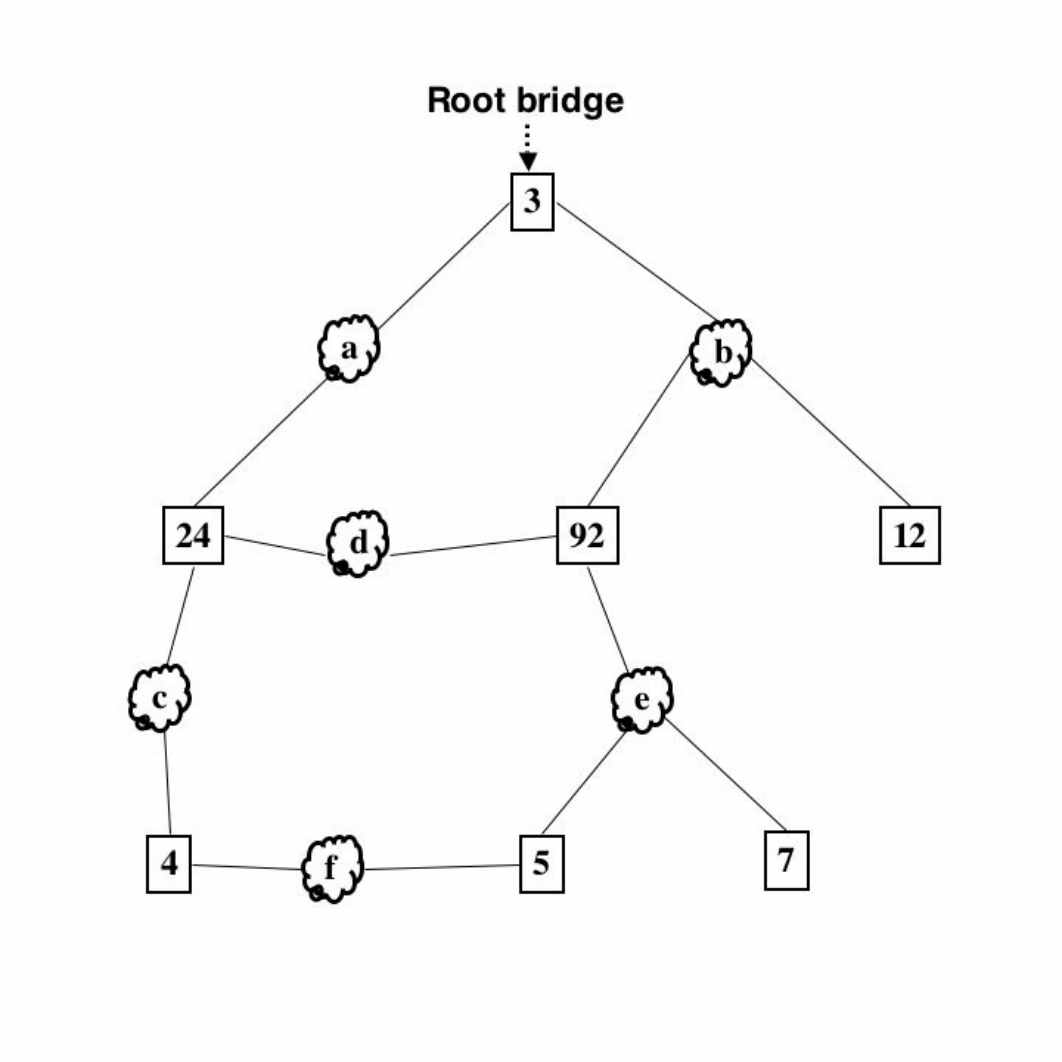
- Spanning Tree Protocol
  - řešení problémů s cykly
  - přepínače si mezi sebou ustaví kostru, po které se data posílají
- Postup
  1. zvolí se kořenový přepínač (nejnižší bridge ID event. MAC adresa)
  2. ustanoví se kostra tak, aby cesta ke kořenovému přepínači byla co nejkratší
     - root port (ke kořenovému přepínači)
     - designated port (do každé sítě)
  3. vypnutí ostatních portů
  4. úprava nerozhodných případů
     - porovnává se bridge ID (event. port ID)

Spanning Tree Protocol

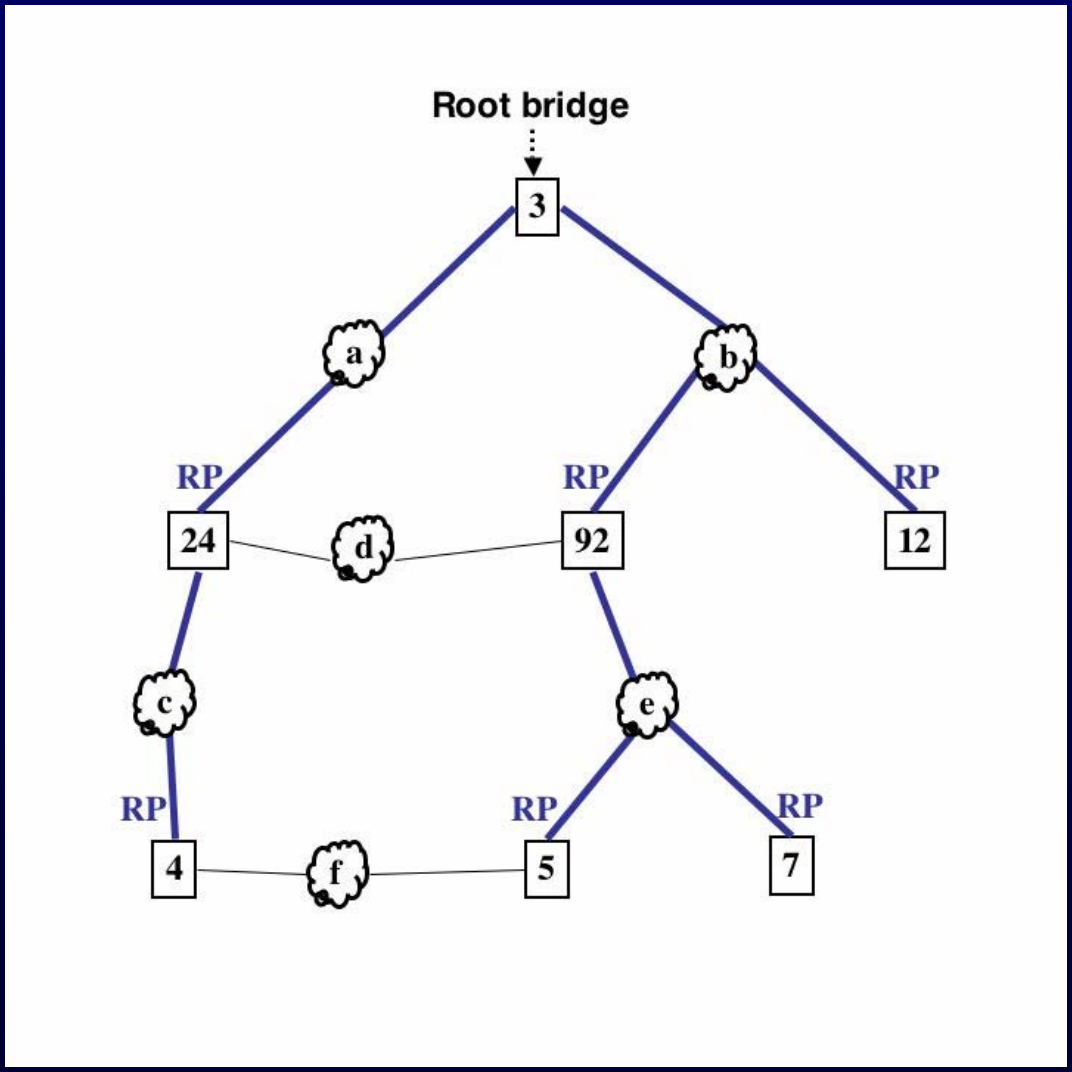
- http://en.wikipedia.org/wiki/Rapid_Spanning_Tree_Protocol



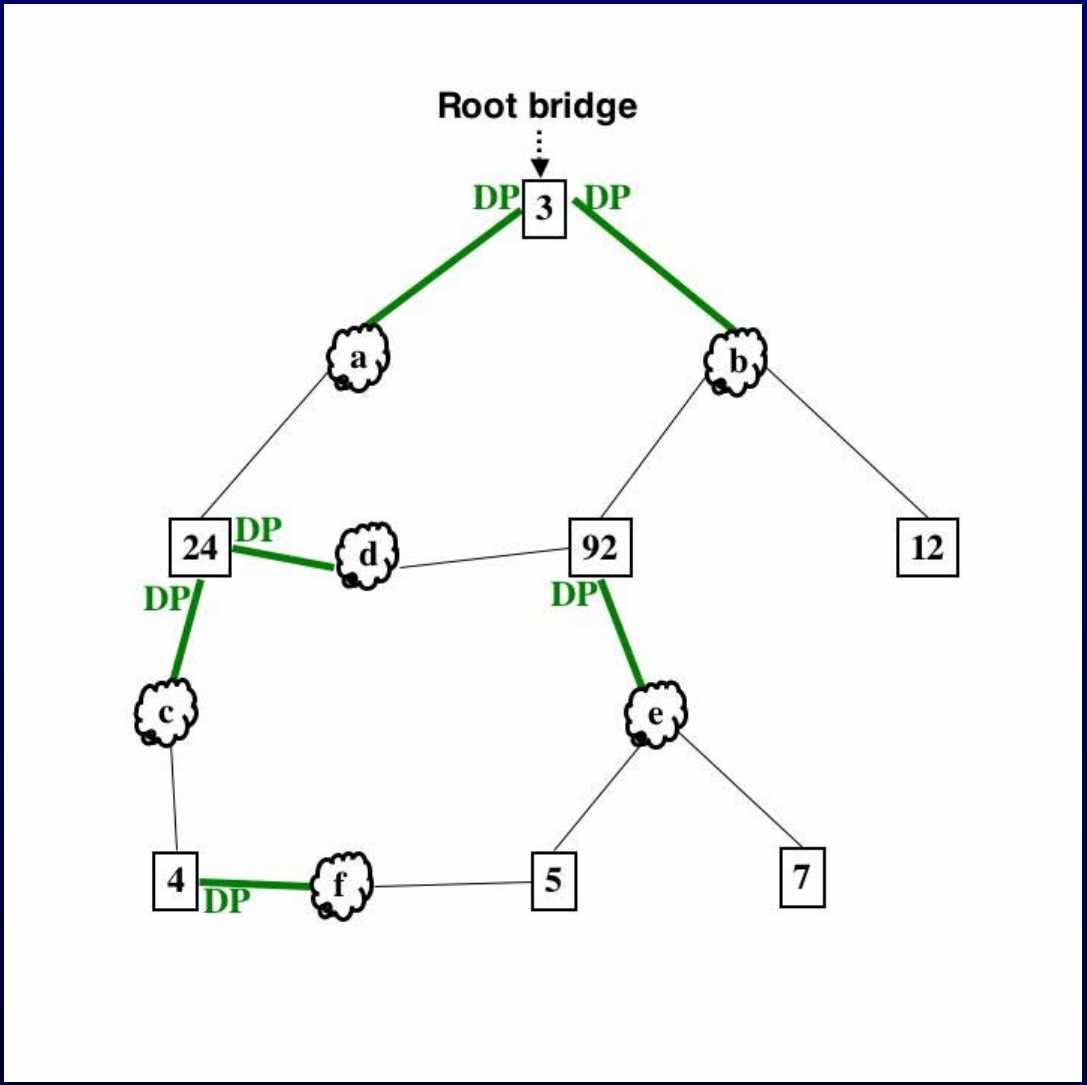
Spanning Tree Protocol



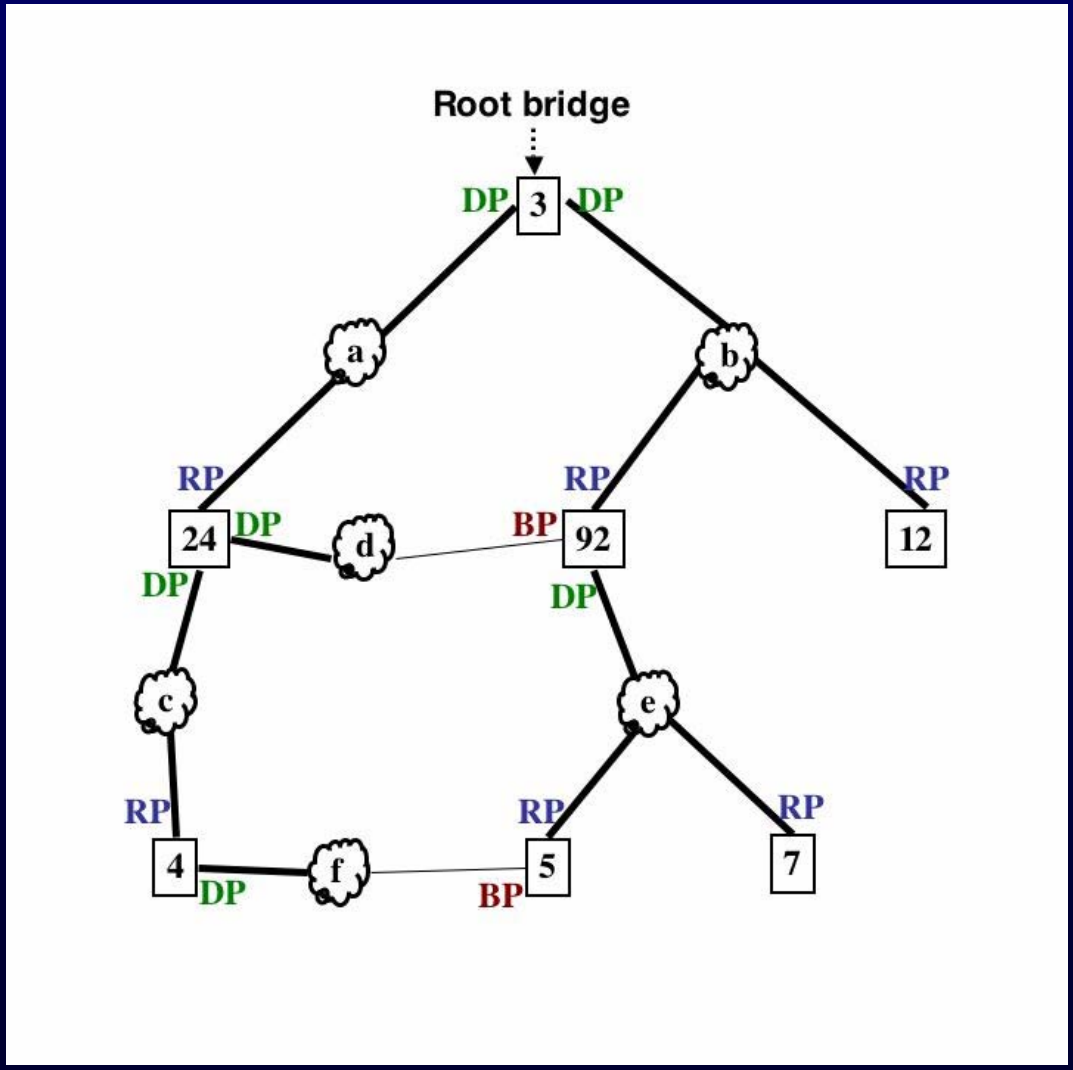
Spanning Tree Protocol



Spanning Tree Protocol



Spanning Tree Protocol



Link State směrování

- Každý směrovač se snaží získat představu o celé síti
  - směrovače si mezi sebou posílají zprávy o topologii svého okolí
    - každý směrovač si otestuje své sousedy
    - zapamatuje si výsledek a pošle jim stavovou zprávu
    - tato informace se propaguje postupně na všechny směrovače (flood)
  - nad sestavenou topologií se pomocí Dijkstrova algoritmu spočítá nejkratší cesta pro každý uzel v síti

Udržování topologie

- link-state algoritmy
- procesy si posílají zprávy st(cislo_procesu, up_pole, casova_znacka)
- pro proces i
- net[k,l] = true iff k-l jsou sousedi a spoj k-l obousměrně funguje

  vp[j] = true iff i-j jsou sousedi a up[j] = true

  ts[j] = maximální časová značka zprávy st(j, ...)

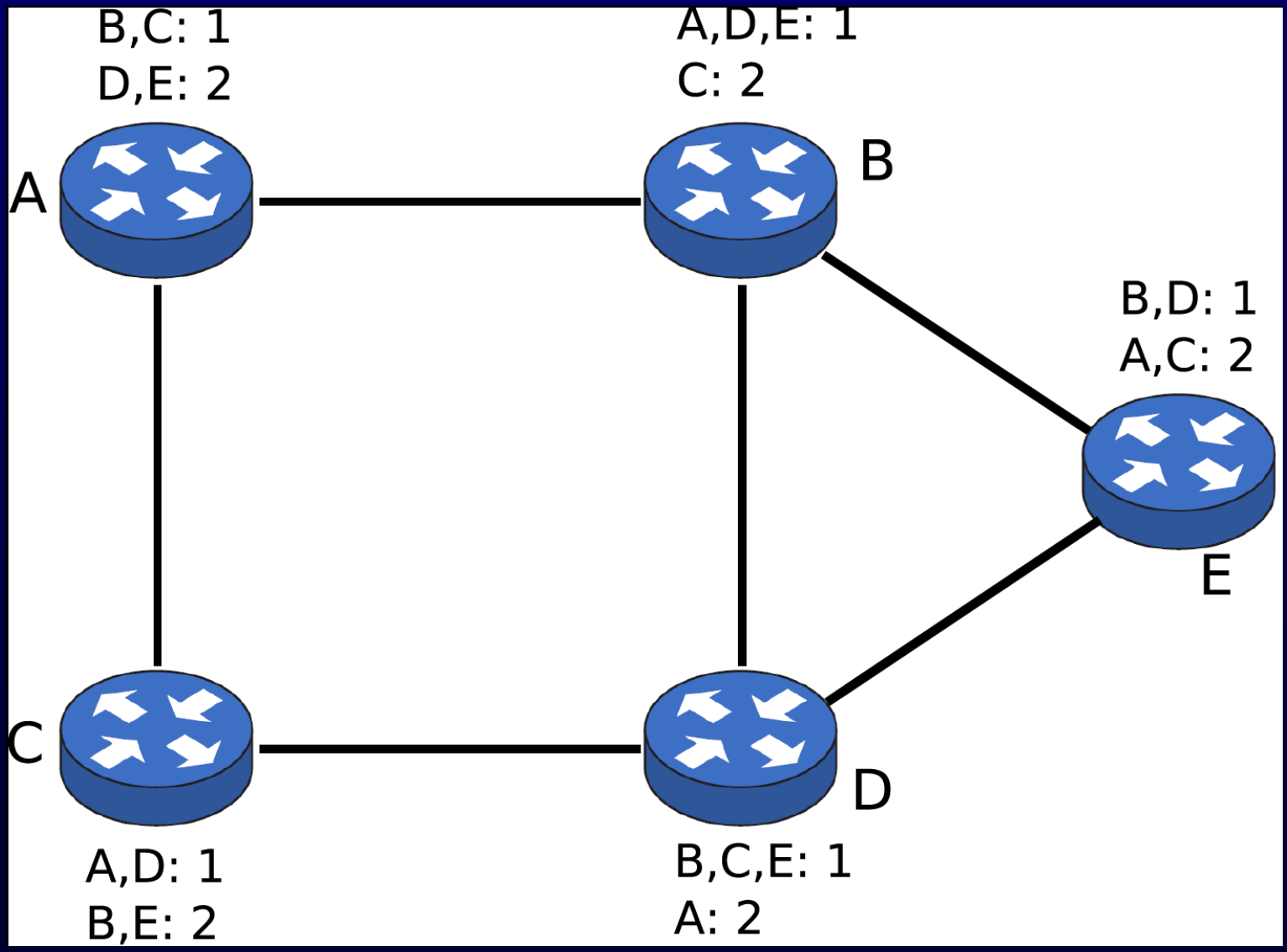
Udržování topologie (2)

```
process p [i: 0..n-1]

inp N    : set { j | p[j] is a neighbor of p[i] },
    up   : array [N] of boolean
var net          : array [0..n-1, 0..n-1] of boolean,
    vp           : array [0..n-1] of boolean,
    ts           : array [0..n-1] of integer,
    f, h : N,
    m            : 0..n,
    k            : 0..n-1,
    t            : integer
par q : N

begin
        true     ->
                 ts[i], m := ts[i]+1, 0;
                 do m<n  ->
                         if (m in N  ∧  up[m])  ->
                                 net[m,i], net[i, m], vp[m] :=
                                         true, true, true
                         □  ~(m in N  ∧  up[m])  ->
                                 net[m,i], net[i, m], vp[m] :=
                                         false, false, false
                         if;  m := m+1
                 od
```

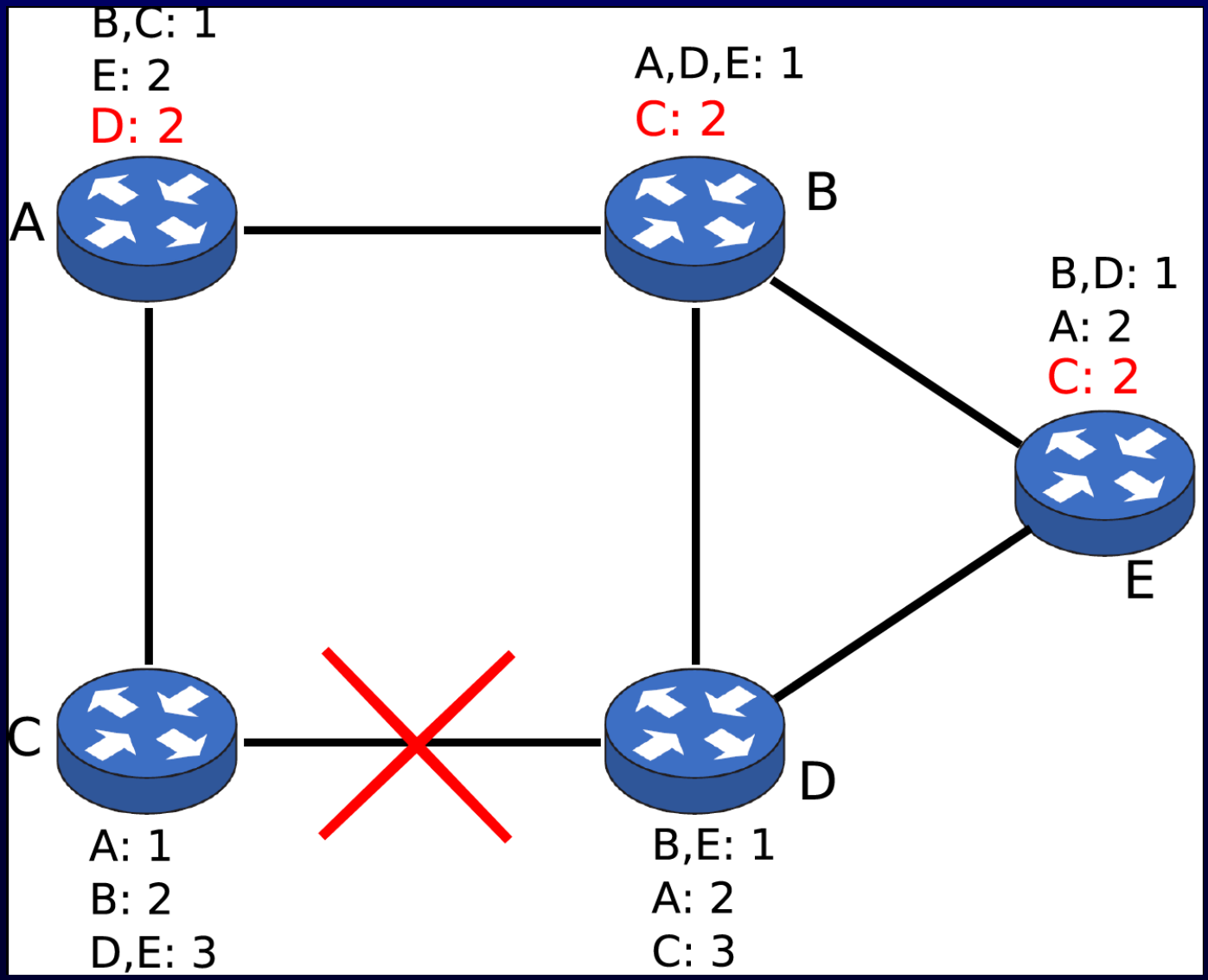
Udržování topologie (3)

```
                    f := random;
                    h := NEXT(N,f);
                    do h/=f  ->
                            if up[h]  ->  send st(i, vp, ts[i]) to p[h]
                            □  ~up[h]  ->  skip
                            fi; h := NEXT(N, h)
                    od
                    if up[f]  ->  send st(i, vp, ts[i]) to p[f]
                    □  ~up[f]  ->  skip
                    fi;
□   rcv st(k, vp, t) from p[g]  ->
          if ts[k] >= t  ->  skip
          □  ts[k] < t  ->
                    ts[k], m := t, 0;
                    do m<n  ->  net[m,k], net[k,m], m := vp[m], vp[m], m+1 od
                    h := NEXT(N, g);
                    do h/=g  ->
                            if up[h]  ->  send st(k, vp, t) to p[h]
                            □  ~up[h]  ->  skip
                            fi; h := NEXT(N, h)
                    od
          fi
□   rcv error from p[g]  ->  skip
end
```

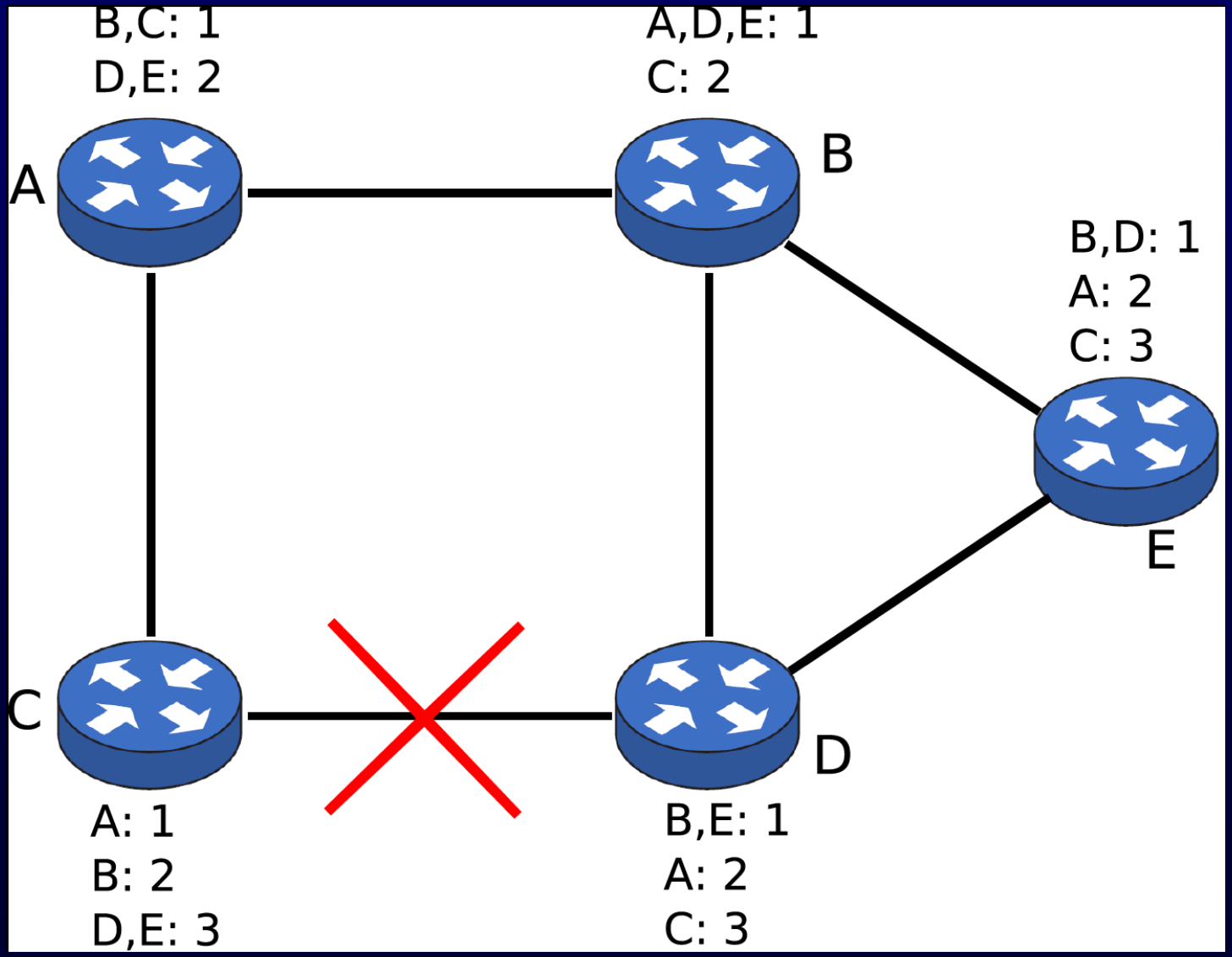
OSPF – chování při výpadku



OSPF – chování při výpadku



OSPF – chování při výpadku



Směrování v Internetu

Základní úrovně směrování v Internetu

- Směrování v podsítích/lokálních sítích
- Směrování v autonomních systémech
- Směrování mezi autonomními systémy
- Páteřní směrování
  - páteř (backbone) je soubor speciálních směrovačů, které znají cestu do každé podsítě na Internetu (v rámci agregace adres)

Směrování v lokální síti

- IP.s - adresa odesílatele
  M.s - maska sítě odesílatele
  IP.d - adresa cíle

```
if (IP.s and M.s) = (IP.d and M.s) ->
        {local delivery}
 (IP.s and M.s) /= (IP.d and M.s) ->
        {use routing table with longest prefix match
        or default gateway}
```

- hledání nejdelšího prefixu ve směrovací tabulce
  - např. hledám 192.168.1.140 v:
    192.168.0.0/16, 192.168.1.0/24, 192.168.1.128/25

Exkurze - rozborka adres sítí

```
-bash-2.05b$ ipcalc 192.168.1.0/25

Address:   192.168.1.0          11000000.10101000.00000001.0 0000000
Netmask:   255.255.255.128 = 25 11111111.11111111.11111111.1 0000000
Wildcard:  0.0.0.127            00000000.00000000.00000000.0 1111111
=>
Network:   192.168.1.0/25       11000000.10101000.00000001.0 0000000 (Class C)
Broadcast: 192.168.1.127        11000000.10101000.00000001.0 1111111
HostMin:   192.168.1.1          11000000.10101000.00000001.0 0000001
HostMax:   192.168.1.126        11000000.10101000.00000001.0 1111110
Hosts/Net: 126                  (Private Internet RFC 1918)


-bash-2.05b$ ipcalc 147.251.51.0/24

Address:   147.251.51.0         10010011.11111011.00110011 .00000000
Netmask:   255.255.255.0 = 24   11111111.11111111.11111111 .00000000
Wildcard:  0.0.0.255            00000000.00000000.00000000 .11111111
=>
Network:   147.251.51.0/24      10010011.11111011.00110011 .00000000 (Class B)
Broadcast: 147.251.51.255       10010011.11111011.00110011 .11111111
HostMin:   147.251.51.1         10010011.11111011.00110011 .00000001
HostMax:   147.251.51.254       10010011.11111011.00110011 .11111110
Hosts/Net: 254
```

Směrování uvnitř AS

- S ... autonomní systém
  r ... směrovač
- směrovač zná
  - IP adresu a masku každé podsítě v S
  - pro každou podsíť *s* v S definuje nejlepší sousední směrovač pro předání
  - pro sítě *t* mimo S
    - explicitní záznamy pro sítě blízké S
    - implicitní záznam pro ostatní

Směrování uvnitř AS (2)

- Rozhodnutí, jestli IP.d leží v S v síti IP.s s maskou M.s

```
if ∃ IP.s = (IP.d and M.s) ->
        {inside S}
□ not (∃ IP.s = (IP.d and M.s)) ->
        {outside S}
```

- Směrovací algoritmy:
  - distance vector - např. RIP
    - distribuovaný směrovací protokol
  - link state - napr. OSPF
    - protokol pro udržování topologie

Směrování mezi AS

- Typy autonomních systémů
  - koncové (stub) AS
  - multihomed AS
  - transit AS
- Autonomous system number (ASN) (RFC1930)
  - 16-bitový identifikátor
  - koncové AS jej nemusí mít přiřazené
  - přiřazuje ICANN (www.icann.org)
    Internet Corporation For Assigned Names and Numbers

Směrování mezi AS – BGP

- Path-vector protocol
  - nevyměňují se pouze ceny cest, ale celé cesty zahrnující všechny skoky
- Pracuje na úrovni sítí, nikoli jednotlivých uzlů/směrovačů
- Základem jsou oznamy (advertisments)
  - zasílají se přes point-to-point spoje
  - oznam obsahuje:
    adresu cílové sítě (CIDR) + atributy cesty (např. atribut path (seznam všech AS na cestě) a identita next-hop směrovače

Proč rozlišovat mezi směrováním uvnitř AS a mezi AS?

- Mezi AS hrají roli mnohem více následující faktory
  - politiky (protože typicky jde o peníze)
  - škálovatelnost (velikosti BGP tabulek)
- Uvnitř AS hraje spíše roli výkon