PA159, přednáška 3

28. 9. a 5. 10. 2007

Co jsme dělali minule...

- Popis AP notace + příklady
- Elementární složky protokolů pro zajištěný přenos
  - typy potvrzování
  - řízení toku (dle příjemce, dle kapacity sítě)
- TCP
  - základní chování
  - problémy na tlustých dlouhých linkách

... a co nás dnes čeká

- Verifikace protokolů v AP notaci
- Směrování
  - elementy směrovacích protokolů
  - distance vector protokoly
  - link state protokoly
- Běžné směrovací protokoly
  - OSPF, BGP

Verifikace protokolů

Verifikace protokolu

- Chceme dokázat žádané chování protokolu
- Minimalistická verifikace programů
  - anotace
    - kdykoli program dosáhne nějaké definované místo v kódu, hodnoty proměnných splňují nějaký predikát
  - ukončení
    - obecně nerozhodnutelné
- Minimalistická verifikace protokolů
  - uzávěra (closure)
  - opakování (recurrence)

Uzávěra (closure)

- stavový predikát *r*
  - vrací *true* nebo *false*
- *r*-stav
  - stav, v němž je hodnota predikátu *r* rovna *true*
- uzávěra
  - stavový predikát *r* je uzávěra iff
    - alespoň jeden stav protokolu P je *r*-stav
    - každý běh protokolu P začínající v *r*-stavu je **nekonečný** a všechny jeho stavy jsou *r*-stavy

Uzávěra (2)

- uzávěra *r*
  - definuje neprázdnou uzavřenou doménu běhu protokolu P
  - doména má alespoň jeden stav vyskytující se v protokolu nekonečněkrát

Opakování (recurrence)

- přechod (b,t)
  - b ... predikát obsahující pouze lokální proměnné procesu P
    t ... akce téhož procesu P
  - (b,t) je **povoleno** ve stavu *r* iff (b $\wedge$ stráž(t)) = true
- redukovaná množina přechodů
  - každý pár různých přechodů má různé akce
  - redukce: $(b_1,t), (b_2,t) \rightarrow (b_1 \vee b_2, t)$

Opakování (2)

- T ... redukovaná množina přechodů v P
  r ... uzávěra v P
- T se opakuje v P iff $\forall$ běhy (stav1; akce1; ...) kde stav1 je r-stav splňují podmínku:
  $\exists(b,t) \in T.\ \forall i.\ \exists j>i.$
  (b,t) je povoleno ve stavu $stav_j$ a $akce_j$ je t
- pokud se T opakuje uvnitř *r* v P a vykonávání P začíná v *r*-stavu, pak alespoň jeden přechod z T bude vykonáván nekonečněkrát

Verifikace

- Vlastnosti protokolu jsou popsány
  - *r* je uzávěra v P
  - T_1 se opakuje uvnitř P

    ...

    T_n se opakuje uvnitř P
- *r* definuje uzavřenou doménu běhu protokolu a T_i definují přechody, z nichž se alespoň jeden vyskytuje nekonečně často
- Tyto 2 vlastnosti potřebujeme ověřit.

Verifikace uzávěry

- Postup důkazu
  1. svědek (witness):
     ukázat, že protokol P má *r*-stav
  2. živost (liveness):
     dokázat, že pro $\forall$ *r*-stavy existuje alespoň jedna povolená akce
     (což garantuje, že r-stav nemůže být koncovým stavem konečného běhu protokolu)
  3. stabilita (stability):
     dokázat, že pokud je v *r*-stavu povolená nějaká akce, pak tato akce vede do *r*-stavu

Verifikace opakování

- Komplement T ≡ ~T
  - pro redukovanou množinu přechodů T
  - ~T = { (~b,t) | ∃ přechod (b,t) v T } ∪ { (true, t) | ∀t, pro něž není přechod v T }
- Ohodnocovací funkce f pro *r*-stavy
  - f∈N pro ∀ *r*-stavy
  - f(s) = k; ∃s→s'; s,s' ... *r*-stavy
    přechod snižuje f iff f(s') < k
    přechod zvyšuje f iff f(s') > k

Verifikace opakování (2)

- Postup důkazu:
  1. žádný návrat (no-retreat):
     je-li v ~T povolen přechod v *r*-stavu procesu P,
     potom jeho provedení nezvýší f
  2. postup (progress):
     pro k > 0:
     ∃ přechod v T, který je povolen v každém *r*-stavu
     kde f=k
     nebo
     ∃ přechod v ~T, který je povolen v každém *r*-stavu
     kde f=k a jeho vykonání snižuje f

Verifikace opakování (2)

- 3. závěr (conclusion)
  dokázat, že pro každý r-stav f=0, ∃ přechod v T, který je povolen
- kombinace 1. a 2. říká, že v r-stavu f>0 buď bude vykonán nějaký přechod v T, nebo bude sníženo f
- kombinace 1. a 3. říká, že pro r-stav f=0 bude vykonán přechod v T
- ohodnocovací funkci je možné rozšířit i na vektor přirozených čísel

Verifikace v případě výskytu chyb

- Typy chyb: přeuspořádání, poškození nebo ztráta dat
  - chyby jsou definované jako akce
- Verifikace zůstává stejná
- Přibývá verifikovat stabilitu pro výskyt chyb
  - pokud je v *r*-stavu definována akce nebo chyba, tak jak pod akcí tak i pod chybou musí protokol P přejít do *r*-stavu

Příklad

- Převrácené kódování Manchester

```
process p

inp data : array [0..1] of integer

var i : integer

begin
        true -> if data[i] = 0 ->     send 0 to r;
                                      send 1 to r
                ▯ data[i] = 1 ->      send 1 to r;
                                      send 0 to r;
                fi;
                i := i + 1;
end
```

Příklad (2)

```
process q

var rcvd : array [0..1] of integer
    j : integer,
    first : boolean,      {first := true}
    b : 0..1

begin
        rcv b from p ->
                if first ->
                        rcvd[j], j, first := b, j+1, false;
                ▯ ~first ->
                        first := true;
                fi;
end
```

Příklad (3)

- Stavový predikát
  r =
  (first $\wedge$ dbl.c $\wedge$ seq.(data, i-1) = seq.(rcvd, j-1); even.c)
  $\vee$
  (~first $\wedge$ ~dbl.c $\wedge$ seq.(data, i-1) = seq.(rcvd, j-1); odd.c)
- c ... obsah ch.p.q
  dbl.c = true iff #ch.p.q = 0 $\vee$ #ch.p.q mod 2 = 0
  dbl.c = false iff #ch.p.q mod 2 = 1

Příklad (3)

- Stavový predikát

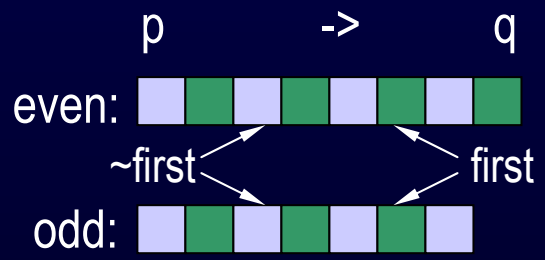
  r =

  (first $\wedge$ dbl.c $\wedge$ seq.(data, i-1) = seq.(rcvd, j-1); even.c)

  $\vee$

  (~first $\wedge$ ~dbl.c $\wedge$ seq.(data, i-1) = seq.(rcvd, j-1); odd.c)

ch.p.q čteno zprava doleva (<-):



c	∅	b2	x;b2;b1
even.c	∅	-	even.x;b1
odd.c	-	b2	odd.x;b2

odd:  ... nemůže nastat, protože odesílání je **atomické**

Příklad (3)

- Stavový predikát
  r =
  (first $\wedge$ dbl.c $\wedge$ seq.(data, i-1) = seq.(rcvd, j-1); even.c)
  $\vee$
  (~first $\wedge$ ~dbl.c $\wedge$ seq.(data, i-1) = seq.(rcvd, j-1); odd.c)
- seq.(y, k-1) = empty
  pro k=0
  seq.(y, k-1) = y[0];...;y[k-1]
  pro k>0

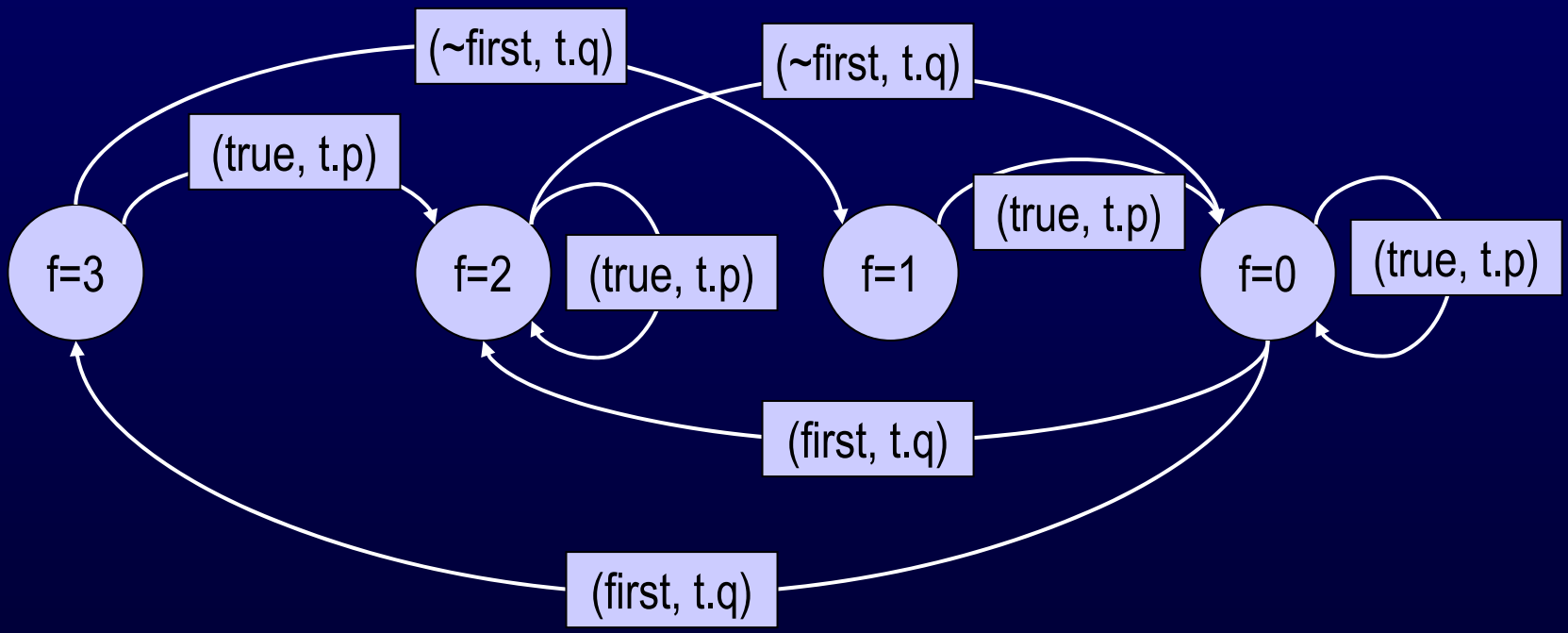
Příklad (4)

- Podmínka svědka (tj. existuje *r*-stav):
  první stav:
  $(i = 0 \wedge \#ch.p.q = 0 \wedge j = 0 \wedge first)$
  ... splňuje
- Podmínka živosti:
  akce v procesu p je vždy aktivována
- Podmínka stability
  proces p má akci, která nemůže negovat r
  proces q vždy splňuje jeden ze dvou termů

Příklad (5)

- Definice T:
  T = { (first, t.q) }
  t.q ... je jediný přechod v q
- Komplement: ~T = {(~first, t.q) , (true, t.p) }
- Ohodnocovací funkce
  f = 0 if r $\wedge$ first $\wedge$ #ch.p.q > 0
  1 if r $\wedge$ first $\wedge$ #ch.p.q = 0
  2 if r $\wedge$ ~first $\wedge$ #ch.p.q > 1
  3 if r $\wedge$ ~first $\wedge$ #ch.p.q = 1

Příklad (6)


(~first, t.q)
(~first, t.q)
(true, t.p)
(true, t.p)
f=3
f=2
(true, t.p)
f=1
f=0
(true, t.p)
(first, t.q)
(first, t.q)

Příklad (6)

- Podmínka žádného návratu:
  splněna - pouze přechody z T zvětšují f
- Podmínka postupu:
  splněna - čtyři přechody nahoře
- Podmínka závěru:
  splněna, dva přechody dole

Elementy směrování

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*
  - každý proces jednoznačně definován trojicí (region, obvod, proces)
- 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

- 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 se všemi hranami ohodnocenými 1
- = distance vector směrování (např. RIP, BGP)

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[h]       -> 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
```

Backward learning routing

```
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
```

Udržování topologie

- 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)

```
                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
```

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 - např. OSPF
    - protokol pro udržování topologie

OSPF verze 2

- Definováno v RFC 2328
- Používá vysílání LS informací a Dijkstrův algoritmus
- Každý směrovač získá kompletní mapu topologie AS (jako orientovaný graf!) a strom cest s nejmenší cenou
- Nastavení ceny spoje (pokud všechny 1, tak je to i strom nejkratších cest)
- Vysílání LS informací
  - číslo vyšší vrstvy v IP hlavičce: 89
  - při změně stavu spoje
  - periodicky minimálně 1x za 30 min.

OSPF verze 2 (2)

- Zabezpečení zpráv mezi OSPF směrovači
- Více cest se stejnou váhou
  - vyvažování zátěže
  - problém s přeuspořádáním paketů
- Směrování multicastu
  - MOSPF (RFC1584)
  - využití znalostí topologie z OSPF pro konstrukci multicastových stromů

OSPF verze 2 (3)

- Hierarchizace jednoho AS
  - oblasti (areas)
  - vysílání LS informací je omezeno na oblast
  - pro směrovače v jiné oblasti AS je oblast black-box
- Hraniční směrovače oblastí
  - jedna oblast uvnitř AS je označena jako páteřní
  - hraniční směrovače náleží do páteřní oblasti a do nejméně jedné další oblasti

OSPF verze 2 (4)

- Informace o ceně mezi oblastmi není přenášena jako informace o ceně spoje, ale jako kumulativní cena cesty
- Páteřní směrovače v AS
  - náleží pouze do páteřní oblasti
- Hraniční směrovače AS
  - používají OSPF (příp. I-BGP) dovnitř AS a BGP-4 na směrování mezi AS

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

BGP-4

- 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

BGP-4 (2)

- 3 základní operace
  - příjem a filtrování oznamů
  - výběr cesty
  - zasílání oznamů
- Příjem a filtrování oznamů
  - peer směrovač hlásí, do jakých AS je schopen doručovat
  - předpokládá se, že peerové nelžou!
  - možnost filtrovat (např. takové cesty, kde AS-PATH je ASN vlastního AS - problém cyklů)

BGP-4 (3)

- Výběr cesty
  - z více možných cest z různých oznamů pro daný AS vybrat jeden, a jeho next-hop směrovač zapsat do směrovacích tabulek
  - způsob výběru definuje směrovací politika (SP)
  - když pro daný AS není nastavena SP, použije se nejkratší cesta
  - www.cisco.com/warp/public/459/25.shtml

BGP-4 (4)

- Zasílání oznamů
  - administrátor definuje politiku oznamů
  - umožňuje kontrolu nad tím, do jakých AS potečou data přes adminův AS
- Politiky peerování
  - neexistují psané standardy
  - u typických komerčních ISP: přes AS tečou pouze ta data, která mají zdroj nebo cíl v daném AS
  - peerovací smlouvy jsou velký business ;-)

BGP-4 (5)

- Komunikace mezi BGP peery
  - potřeba spolehlivé komunikace (na rozdíl od OSPF, které si spolehlivost nad IP řeší samo)
  - TCP port 179
  - 4 typy zpráv: OPEN, UPDATE, KEEPALIVE, NOTIFICATION
- OPEN
  - ustavení spojení mezi BGP peery
  - obsahuje typicky autentizační informace
  - pozitivní odpověd na OPEN je KEEPALIVE

BGP-4 (6)

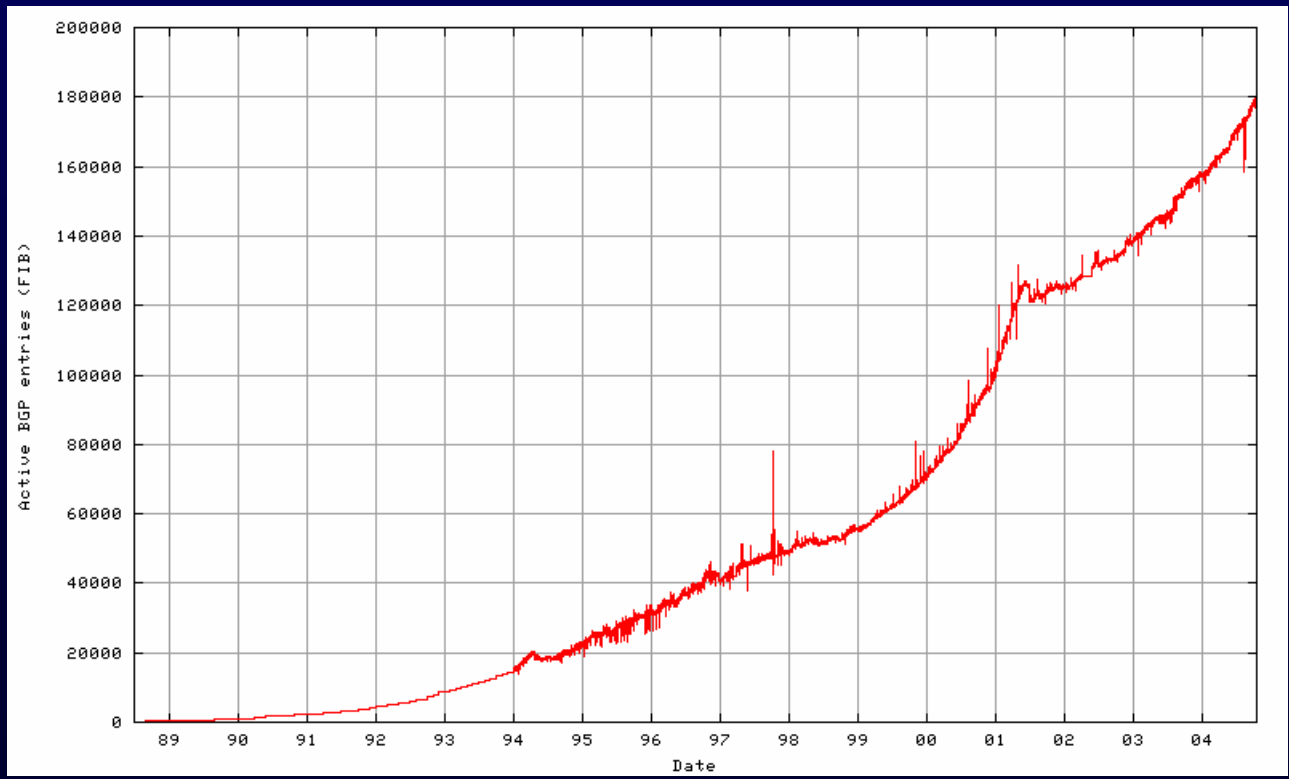
- UPDATE
  - šíření oznamů o cestách
    - oznámení, že lze přeze mne směrovat do daného AS
    - stažení takového oznamu (oznam je platný až do jeho explicitního stažení!)
- KEEPALIVE
  - positivní odpověd na OPEN
  - udržování spojení
- NOTIFICATION
  - oznámení o chybě nebo o ukončení spojení

I-BGP

- Distribuce informací o přilehlých AS mezi směrovači uvnitř AS
  - všechny směrovače uvnitř AS se považují za peery z pohledu I-BGP
  - I-BGP směrovače mohou oznamovat pouze cesty, které se dozvědely přímo od jiného I-BGP směrovače

Velkosti BGP tabulek

- Ukázka velikosti BGP tabulek: http://bgp.potaroo.net/



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