Vypracované otázky k MSZ pro rok 2022

Specializace NNET

30. dubna 2022

Vladimír Dušek, xdusek27

Specializace Počítačové sítě – NNET

1. Architektura superskalárních procesorů a algoritmy zpracování instrukcí mimo pořadí, predikce skoků.
2. Paměťová konzistence a předbíhání operací čtení a zápisu, podpora virtuálního adresového prostoru.
3. Datový paralelismus SIMD, HW implementace a SW podpora.
4. Architektury se sdílenou pamětí UMA a NUMA, zajištění lokality dat.
5. Problém koherence pamětí cache na systémech se sdílenou pamětí, protokol MSI.
6. Paralelní zpracování v OpenMP: Smyčky, sekce a tasky a synchronizační prostředky.
7. Pravděpodobnost, podmíněná pravděpodobnost, nezávislost.
8. Náhodná proměnná, typy náhodné proměnná, funkční a číselné charakteristiky, významná rozdělení pravděpodobnosti.
9. Bodové a intervalové odhady parametrů, testování hypotéz o parametrech.
10. Vícevýběrové testy, testy o rozdělení, testy dobré shody.
11. Regresní analýza.
12. Markovské řetězce a základní techniky pro jejich analýzu.
13. Randomizované algoritmy (Monte Carlo a Las Vegas algoritmy).
14. Problém generalizace strojového učení a přístup k jeho řešení (trénovací, validační a testovací sada, regularizace, předtrénování, multi-task learning, augmentace dat, dropout, ...)
15. Generativní modely a diskriminativní přístup ke klasifikaci (gaussovský klasifikátor, logistická regrese, ...)
16. Neuronové sítě a jejich trénování (metoda gradientního sestupu, účelová (loss) funkce, výpočetní graf, aktivační funkce, zápis pomocí maticového násobení, ...)
17. Neuronové sítě pro strukturovaná data (konvoluční a rekurentní sítě, motivace, základní vlastnosti, použití)
18. Prohledávání stavového prostoru (informované a neinformované metody, lokální prohledávání, prohledávání v nejistém prostředí, hraní her, CSP úlohy)
19. Klasifikace formálních jazyků (Chomského hierarchie), vlastnosti formálních jazyků a jejich rozhodnutelnost.
20. Konečné automaty (jazyky přijímané KA, varianty KA, minimalizace KA, Mihill-Nerodova věta).
21. Regulární množiny, regulární výrazy a rovnice nad regulárními výrazy.
22. Zásobníkové automaty (jazyky přijímané ZA, varianty ZA).
23. Turingovy stroje (jazyky přijímané TS, varianty TS, lineárně omezené automaty, vyčíslitelné funkce).
24. Nerozhodnutelnost (problém zastavení TS, princip diagonalizace a redukce, Postův korespondenční problém).
25. Časová a paměťová složitost (třídy složitosti, úplnost, SAT problém).
26. Postrelační a rozšířené relační databáze (objektový a objektově relační databázový model – struktura a operace; podpora práce s XML a JSON dokumenty v databázích).
27. NoSQL databáze (porovnání relačních a NoSQL; CAP věta a ACID/BASE principy; typy NoSQL databází; dotazování v NoSQL databázích; agregace dat pomocí Map-Reduce a agregační pipeline).
28. Získávání znalostí z dat (pojem znalost; typické zdroje dat; základní úlohy získávání znalostí; analytické projekty a proces získávání znalostí z dat).

29. Porozumění datům (důvod a cíl; popisné charakteristiky dat a vizualizační techniky; korelační analýza).
30. Prostorové DB (problematika mapování prostoru, ukládání, indexace; využití).
31. Indexace (nejen) v prostorových DB (kD-Tree a Grid File (a jejich varianty), R-Tree).
32. Lambda kalkul (definice všech pojmů, operací...).
33. Práce v lambda kalkulu (demonstrace reprezentace čísel a pravdivostních hodnot a operací nad nimi).
34. Haskell – lazy evaluation (typy v jazyce včetně akcí, uživatelské typy, význam typových tříd, demonstrace lazy evaluation).
35. Prolog – způsob vyhodnocení (základní princip, unifikace, chování vestavěných predikátů, operátor řezu – vhodné a nevhodné užití).
36. Prolog – změna DB/programu za běhu (demonstrace na prohledávání stavového prostoru, práce se seznamy).
37. Model PRAM, suma prefixů a její aplikace.
38. Distribuované a paralelní algoritmy – algoritmy nad seznamy, stromy a grafy.
39. Interakce mezi procesy a typické problémy paralelismu (synchronizační a komunikační mechanismy).
40. Distribuované a paralelní algoritmy – předávání zpráv a knihovny pro paralelní zpracování (MPI).
41. Distribuovaný broadcast, synchronizace v distribuovaných systémech.
42. Klasifikace a vlastnosti paralelních a distribuovaných architektur, základní typy jejich topologií.
43. Distribuované a paralelní algoritmy – algoritmy řazení, select, algoritmy vyhledávání.
44. Bezdrátové lokální sítě (Wifi, Bluetooth).
45. Hledání minimální kostry obyčejného grafu (pojmy, stromy a kostry, Kruskalův algoritmus, Primův algoritmus).
46. Hledání nejkratších cest ze zdrojového uzlu do všech ostatních uzlů grafu (Bellman-Fordův algoritmus, Dijkstrův algoritmus).
47. Klasifikace algoritmů volby koordinátora, algoritmus Bully a jeho složitost.
48. Podmínky konsistentního globálního stavu distribuovaného systému.
49. Principy distribuovaného zpracování MapReduce, průběh a jednotlivé operace distribuovaného výpočtu pomocí MapReduce, jeho implementace v Apache Hadoop a Apache Spark.
50. Symetrická kryptografie. Vlastnosti, vlastnosti bezpečného algoritmu, délka klíče, útok silou, příklady symetrických algoritmů, Feistelovy šifry, DES, režimy činnosti, proudové šifry.
51. Asymetrická kryptografie, vlastnosti, způsoby použití, poskytované bezpečnostní funkce, elektronický podpis a jeho vlastnosti, hybridní kryptografie, algoritmus RSA, generování klíčů, šifrování, dešifrování.
52. Hašovací funkce, klíčovaný haš a MAC a jejich použití a vlastnosti.
53. Správa klíčů v asymetrické kryptografii (certifikáty X.509).
54. Základní architektury přepínačů, algoritmy pro plánování, řešení blokování, vícestupňové přepínací sítě.
55. Základní funkce směrovače, zpracování paketů ve směrovači, typy přepínání a architektur.
56. Metody pro výpočet směrování v sítích (Bellman-Ford, Dijkstra, Path vector, DUAL).
57. Řízení toku dat (flow-control) a prevence zahlcení (congestion-control) na transportní vrstvě (MP-TCP, QUIC, SCTP, DCCP).
58. Metody detekce síťových incidentů (signatury, statistické metody) a nástroje (IDS/IPS).
59. Sítě Peer-to-Peer: vlastnosti, chování, způsoby směrování. Strukturované a nestrukturované sítě.

60. Události v JavaScriptu (smyčka událostí, asynchronní programování, klientské události, obsluha událostí)
61. Přenos a distribuce webových dat (URI, protokol HTTP, proudy HTTP, CDN, XHR)
62. Bezpečnost webových aplikací (SOP, XSS, CSRF, bezpečnostní hlavičky HTTP)

Obsah

1 BMS – Bezdrátové lokální sítě (Wifi, Bluetooth). 5

2 GAL – Hledání minimální kostry obyčejného grafu (pojmy, stromy a kostry, Kruskalův algoritmus, Primův algoritmus). 6

3 GAL – Hledání nejkratších cest ze zdrojového uzlu do všech ostatních uzlů grafu (Bellman-Fordův algoritmus, Dijkstrův algoritmus). 17

4 PDI – Klasifikace algoritmů volby koordinátora, algoritmus Bully a jeho složitost. 26

5 PDI – Podmínky konsistentního globálního stavu distribuovaného systému. 32

6 PDI – Principy distribuovaného zpracování MapReduce, průběh a jednotlivé operace distribuovaného výpočtu pomocí MapReduce, jeho implementace v Apache Hadoop a Apache Spark. 36

7 KRY – Symetrická kryptografie. Vlastnosti, vlastnosti bezpečného algoritmu, délka klíče, útok silou, příklady symetrických algoritmů, Feistelovy šifry, DES, režimy činnosti, proudové šifry. 43

8 KRY – Asymetrická kryptografie, vlastnosti, způsoby použití, poskytované bezpečnostní funkce, elektronický podpis a jeho vlastnosti, hybridní kryptografie, algoritmus RSA, generování klíčů, šifrování, dešifrování. 57

9 KRY – Hašovací funkce, klíčovaný haš a MAC a jejich použití a vlastnosti. 63

10 KRY – Správa klíčů v asymetrické kryptografii (certifikáty X.509). 70

11 PDS – Základní architektury přepínačů, algoritmy pro plánování, řešení blokování, vícestupňové přepínací sítě. 76

12 PDS – Základní funkce směrovače, zpracování paketů ve směrovači, typy přepínání a architektur. 93

13 PDS – Řízení toku dat (flow-control) a prevence zahlcení (congestion-control) na transportní vrstvě (MP-TCP, QUIC, SCTP, DCCP). 94

Kapitola 1

BMS – Bezdrátové lokální sítě (Wifi, Bluetooth).

1.1 Metadata

- Předmět: Bezdrátové a mobilní sítě (BMS)
- Přednáška:
  - [[**todo**]]
- Záznam:
  - [[**todo**]]

1.2 Úvod a kontext

[[**todo**]]

Kapitola 2

GAL – Hledání minimální kostry obyčejného grafu (pojmy, stromy a kostry, Kruskalův algoritmus, Primův algoritmus).

2.1 Metadata

- Předmět: Grafové algoritmy (GAL)
- Přednáška:
  - 5) Stromy, minimální kostry, Jarníkův a Borůvkův algoritmus.
  - 6) Růst minimální kostry, algoritmy Kruskala a Prima.
- Záznam:
  - 2020-10-22
  - 2020-10-29

2.2 Úvod a kontext

Orientovaný graf Orientovaný graf je dvojice $G = (V, E)$, kde V je konečná množina uzlů a $E \subseteq V \times V$ je množina hran.

Neorientovaný graf Neorientovaný graf je dvojice $G = (V, E)$, kde V je konečná množina uzlů a $E \subseteq \binom{V}{2}$ je množina hran. (Hrana je tedy dvouprvková množina, avšak běžně se držíme stejného značení jako u orientovaných grafů a používáme dvojici.)

Ohodnocený graf Ohodnocený graf je takový graf, jehož každá hrana má přiřazenou nějakou hodnotu, typicky definovanou pomocí váhové funkce $w : E \mapsto \mathbb{R}$.

Podgraf Graf $G' = (V', E')$ je podgraf grafu $G = (V, E)$ jestliže $V' \subseteq V$ a $E' \subseteq E$.

Sled Posloupnost uzlů $\langle v_0, v_1, \ldots, v_k \rangle$, kde $(v_{i-1}, v_i) \in E$ pro $i = 1, \ldots, k$ se nazývá sled délky k z v_0 do v_k.

Uzavřený sled Sled $\langle v_0, v_1, \ldots, v_k \rangle$ se nazývá uzavřený, pokud existuje hrana (v_0, v_k).

Dosažitelnost Pokud existuje sled s z uzlu u do uzlu v, říkáme, že v je dosažitelný z u sledem s, značeno $u \overset{s}{\Rightarrow} v$.

Tah Tah je sled ve kterém se neopakují hrany.

Cesta Cesta je sled ve kterém se neopakují uzly.

Souvislý graf Neorientovaný graf se nazývá souvislý, pokud mezi libovolnými dvěma uzly existuje cesta.

Kružnice Uzavřená cesta se nazývá kružnice.

Cyklus Orientovaná kružnice se nazývá cyklus (první a poslední uzel je shodný).

Prostý graf Orientovaný graf bez cyklů se nazývá prostý.

Acyklický graf Graf je bez cyklů, resp. kružnic, se nazývá acyklický.

Strom Graf, který je souvislý a acyklický, se nazývá strom.

Kostra Strom, který tvoří podgraf souvislého grafu na množině všech jeho vrcholů, se nazývá kostra (*spanning tree*).

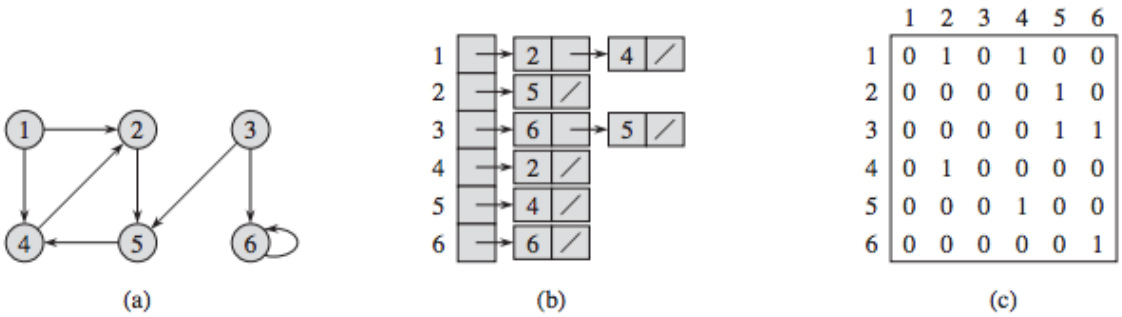
Minimální kostra Nechť $G = (V, E)$ je souvislý neorientovaný graf s váhovou funkcí $w : E \mapsto \mathbb{R}$. Minimální kostra (*MST, minimum spanning tree*) je strom $G' = (V, E')$, kde $E' \subseteq E$ a

$$w(E') = \sum_{(u,v) \in T} w(u, v)$$

je minimální ze všech možných alternativních koster.

Seznam sousedů Seznam sousedů (*Adj*, *adjacency list*) je reprezentace grafu v paměti. Jde o preferovanou variantu pro řídké grafy – kde $m << n^2$. Pro každý uzel máme definovaný seznam jeho sousedů.

Matice sousednosti Matice sousednosti (*adjacency matrix*) je reprezentace grafu v paměti. Jde o preferovanou variantu pro husté grafy – kde m je skoro n^2.



Obrázek 2.1: Příklad reprezentace grafu pomocí seznamu sousedů a matice sousednosti.

2.3 Generický algoritmus

Hledání minimální kostry je problém, který lze řešit algoritmy, které spadají do kategorie tzv. hladových (*greedy*) deterministických algoritmů. Spočívají v tom, že průběžně odhadují kostru přidáváním dalších hran a nikdy se nemusejí vracet (neprovádí se *backtracking*). Generický algoritmus tvoří jakousi základní kostru pro další, už konkrétní, algoritmy.

Řez Nechť $G = (V, E)$ je graf. Řez grafu G je dvojice $(S, V - S)$, kde $\emptyset \subseteq S \subseteq V$.

Křížení Hrana $(u, v) \in E$ kříží řez $(S, V - S)$, pokud jeden její konec je v S a druhý v $V - S$.

Respektování Nechť $A \subseteq E$ je množina hran. Řez $(S, V - S)$ respektuje množinu hran A, pokud žádná hrana v A nekříží řez $(S, V - S)$.

Lehkost Nechť $(S, V - S)$ je řez a B je množina hran, která ho kříží. Hrana z množiny B s nejmenší hodnotou se nazývá lehká.

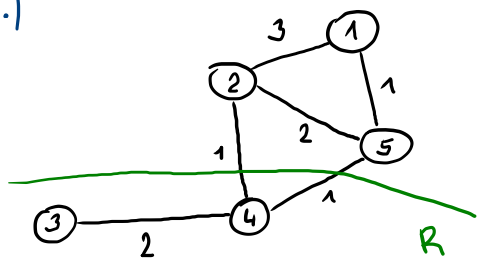
Bezpečnost Nechť $G = (V, E)$ je souvislý neorientovaný graf s reálnou váhovou funkcí w. Nechť $A \subseteq E$ je součástí nějaké minimální kostry G. Nechť $(S, V - S)$ je řez, který respektuje A. Nechť (u, v) je lehká hrana křížící $(S, V - S)$. Pak hrana (u, v) je bezpečná pro A.

```
def generic_mst(G):
    # G je graf
    A~= {}# A~je mnozina hran rozpracovane minimalni kostry
    while netvori_kostru(A, G):
        for hrana in G.E:
            if je_bezpecna(A, hrana):
                A~+= {hrana}
    return
A~
```

Výpis 2.1: Generický algoritmus. Před každou iterací algoritmu je množina A podmnožinou nějaké minimální kostry. Hrana $(u, v) \in E$ je bezpečná pro A, pokud $A \cup \{(u, v)\}$ je podmnožinou nějaké minimální kostry.

Př. $G = (V, E)$ $\quad W: E \to \mathbb{R}$ (definované váhami)

1.)



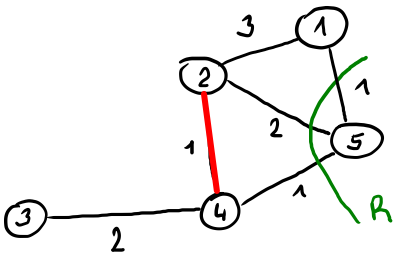
$A = \{\}$

$R = (\{1,2,5\}, \{3,4\})$

$LH = \{(2,4), (4,5)\}$

$A \leftarrow A \cup \{(2,4)\}$

2.)



$A = \{(2,4)\}$

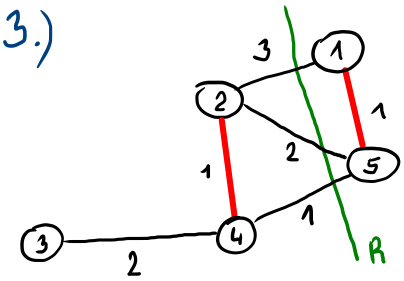
$R = (\{1,2,3,4\}, \{5\})$

$LH = \{(1,5), (4,5)\}$

$A \leftarrow A \cup \{(1,5)\}$

Obrázek 2.2: Příklad, část 1.

3.)



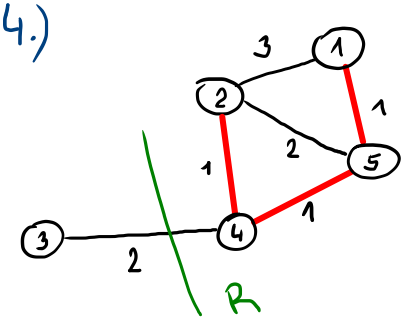
$A = \{(1,5), (2,4)\}$

$R = (\{1,5\}, \{2,3,4\})$

$LH = \{(4,5)\}$

$A \leftarrow A \cup \{(4,5)\}$

4.)



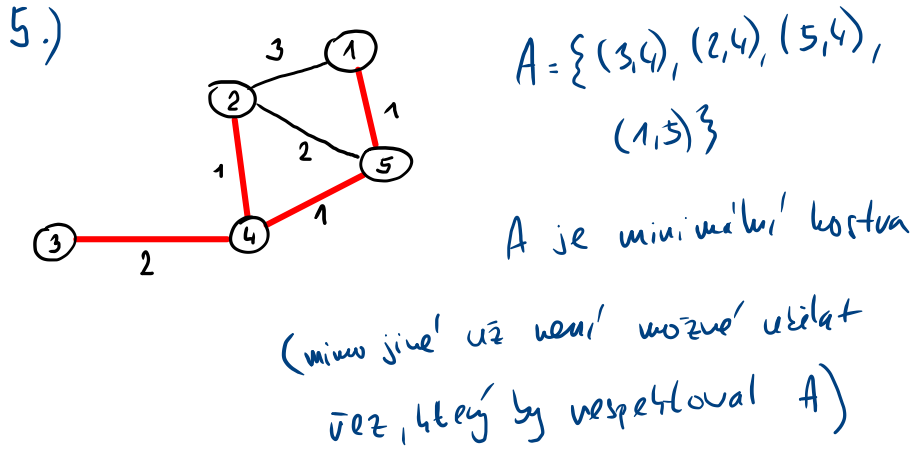
$A = \{(1,5), (2,4), (4,5)\}$

$R = (\{1,2,4,5\}, \{3\})$

$LH = \{(3,4)\}$

$A \leftarrow A \cup \{(3,4)\}$

Obrázek 2.3: Příklad, část 2.



Obrázek 2.4: Příklad, část 3.

2.4 Kruskalův algoritmus

Kruskalův a Primův algoritmus se liší v tom, jakým způsobem vybírají bezpečnou hranu. Kruskalův algoritmus nahlíží na A jako na les a hledá hranu s nejmenším ohodnocením, která spojuje stromy v lese. Na konci je A jeden strom.

```
1  def kruskal_mst(G):
2      # G je graf
3
4      # inicializace, kazdy uzel je ve sve mnozine
5      A~= {}# A~je mnozina hran rozpracovane minimalni kostry
6      for v~in G.V:
7          make_set(v)
8
9      # seradit vzestupne podle w
10     E = sort(G.E, G.w)
11
12     for (u, v) in E:
13         if find_set(u) != find_set(v):
14             A~+= {(u, v)}
15             union(u, v)
16
17     return
18 A~
```

Výpis 2.2: Kruskalův algoritmus. Funkce `make_set(v)` vytvoří množinu obsahující v, `find_set(v)` vrátí reprezentanta množiny ve které se nachází v, `union(u, v)` sjednotí dvě množiny obsahující u a v.

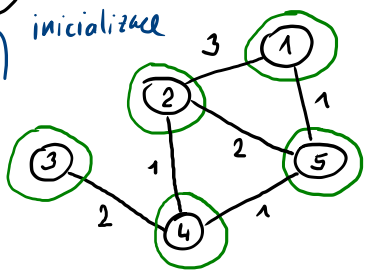
2.4.1 Složitost

- Řádek 5 – $O(1)$
- Řádek 6-7 – n-krát složitost *make_set* (n je počet uzlů).
- Řádek 10 – $O(m \cdot \log(m))$ (m je počet hran).
- Řádky 12-15 – Závisí na implementaci $find_set$ a $union$.
  - Při implementaci seznamem s heuristickou celkem: $O(m + n \cdot \log(n))$.
  - Při stromové implementaci s váhami a zkratkami celkem: $O((m + n) \cdot \alpha(n))$. Kde α je velmi pomalu rostoucí funkce ($\alpha \leq 4$).
- Pro souvislý graf platí $m > n$. Proto množinové operace stojí $O(m \cdot \alpha(n))$. Jelikož $\alpha(n) = O(\log(n)) = O(\log(m))$, tak celková složitost je $O(m \cdot \log(m))$.
- Dále platí $m < n^2$, pak $\log(m) = O(\log(n))$, proto celkem: $O(m \cdot \log(n))$.

2.4.2 Příklad

Př.

0.) inicializace



$A = \{\}$

make_sets

$E = [(1,3), (4,5), (2,4), (2,3), (3,4), (1,2)]$

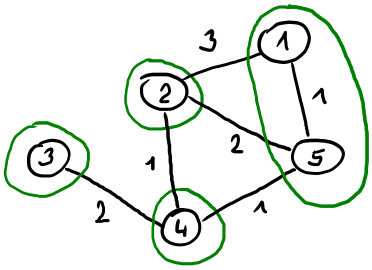
1.) $E = [(4,5), (2,4), (2,3), (3,4), (1,2)]$

$(1,5)$

sets jsou různé

$A \leftarrow A \cup \{(1,5)\}$

union $(1,5)$



Obrázek 2.5: Příklad, část 1.

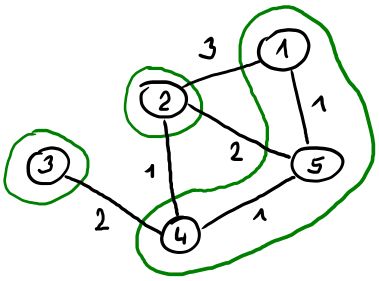
2.) $E = [(2,4), (2,3), (3,4), (1,2)]$

$(4,5)$

sets jsou různé

$A \leftarrow A \cup \{(4,5)\}$

union $(4,5)$



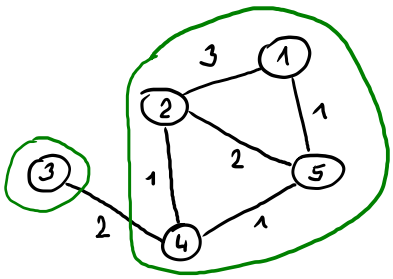
3.) $E = [(2,3), (3,4), (1,2)]$

$(2,4)$

sets jsou různé

$A \leftarrow A \cup \{(2,4)\}$

union $(2,4)$

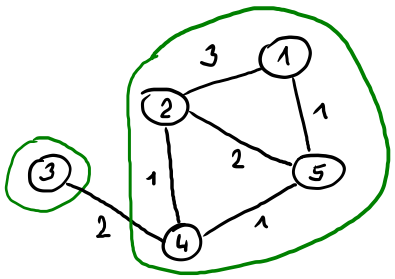


Obrázek 2.6: Příklad, část 2.

4.) $E = [(3,4), (1,2)]$

$(2,5)$

sets nejsou různé



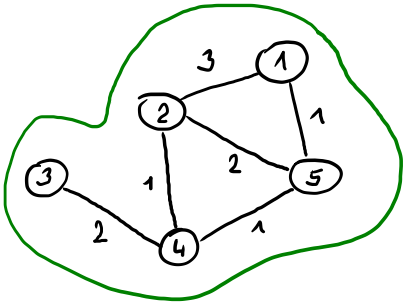
4.) $E = [(1,2)]$

$(3,4)$

sets jsou různé

$A \leftarrow A \cup \{(3,4)\}$

union$(3,4)$



5.) $E = [\ ]$, $(1,2)$, sets nejsou různé

Obrázek 2.7: Příklad, část 3.

2.5 Primův-Jarníkův algoritmus

Primův algoritmus buduje tzv. *A* strom. Má zadaný určitý uzel, ze kterého hledá nejbližší další uzel, který by připojil. A pak další a další.

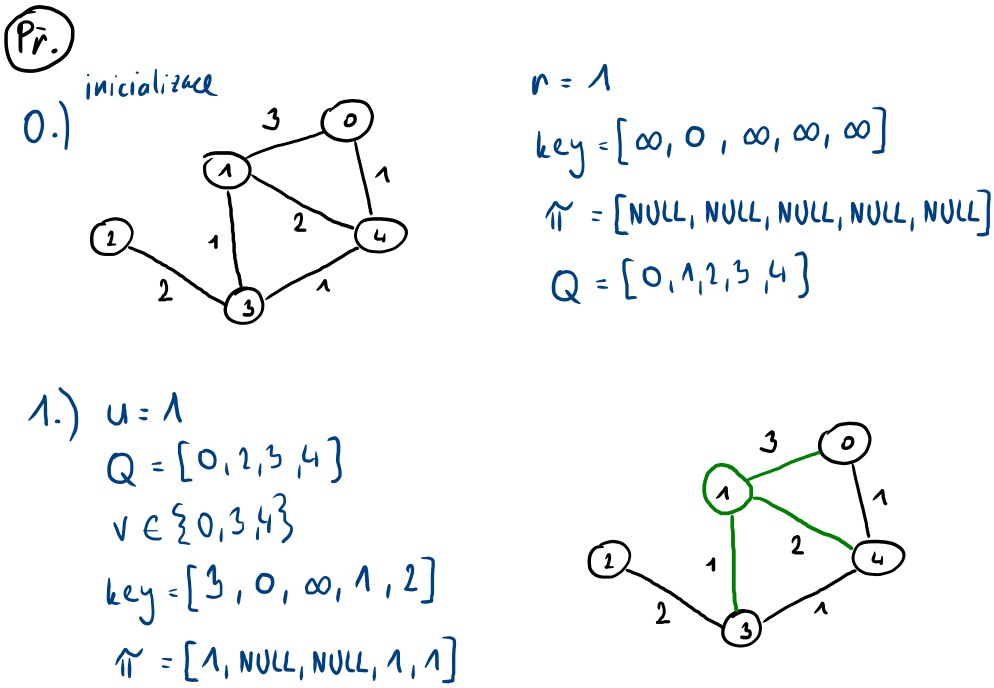
```
1  def prim_mst(G, r):
2      # G je graf
3      # r je vychozi uzel
4
5      for u~in G.V:
6          key[u] = INF # pole cen prechodu, kolik stoji prechod do vrcholu na indexu
7          pi[u] = NULL # pole predchudcu, kdo je predchudce vrcholu na indexu
8
9      key[r] = 0
10     Q = Queue(G.V) # prioritni fronta uzlu
11
12     while not Q.empty():
13         u~= Q.extract_min(key) # vrati prvek z~Q s~nejmensi hodnotou v~key
14
15         # pro vsechny sousedy uzlu u~(Adj je seznam sousedu)
16         for v~in Adj[u]:
17             # pokud je levnejsi cesta a jeste to neni prozkoumany uzel
18             if v~in Q and w(u, v) < key(v):
19                 pi[v] =
20 u~key[v] = w(u, v)
21                 Q.decrease_key(key) # aktualizace prioritni fronty
22
23     return pi
```

Výpis 2.3: Primův algoritmus.

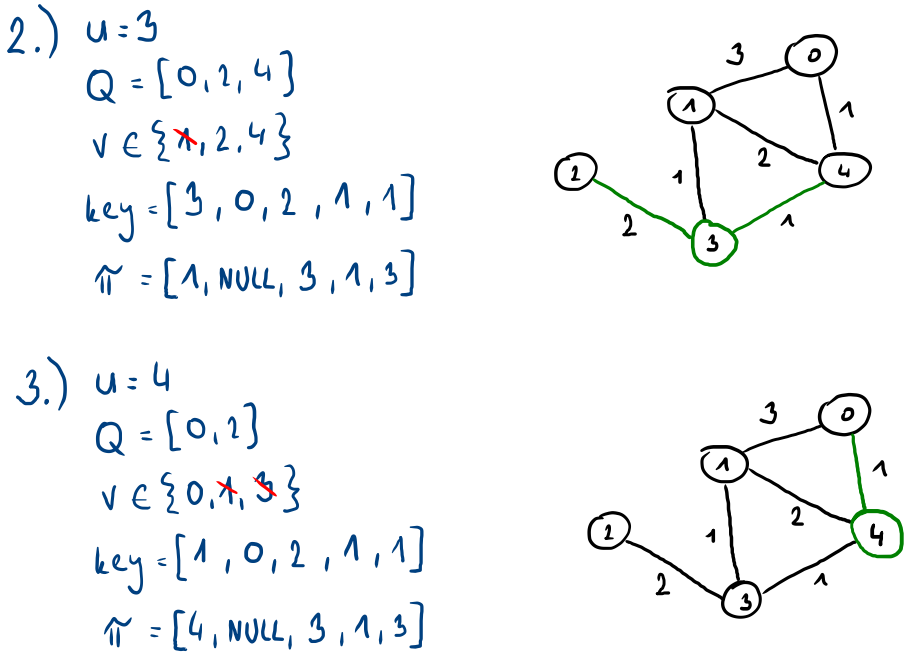
2.5.1 Složitost

- Řádky 5-10 – $O(n)$ za použití binární haldy (n je počet uzlů).
- Řádky 12-13 – While cyklus se provede n-krát a protože *extract_min* stojí $O(\log(n))$, tak je celková složitost $O(n \cdot \log(n))$.
- Řádek 16 – For cyklus se provede $O(m)$ krát, protože dálka všech seznamů sousedů je dohromady $2m$ (m je počet hran).
- Řádek 18-20 – $O(1)$.
- Řádek 21 – $O(\log(n))$.
- Jelikož $m > n$, tak celkem $O(n \cdot \log(n) + m \cdot \log(n)) = O(m \cdot \log(n))$.

2.5.2 Příklad



Obrázek 2.8: Příklad, část 1.



Obrázek 2.9: Příklad, část 2.

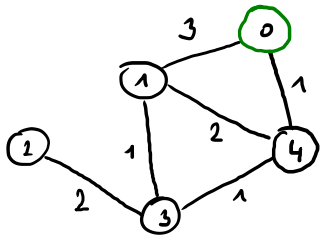
4.) $u = 0$

$Q = [2]$

$v \in \{\cancel{1}, \cancel{4}\}$

$key = [1, 0, 2, 1, 1]$

$\pi = [4, NULL, 3, 1, 3]$



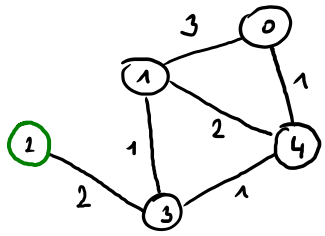
5.) $u = 2$

$Q = [\ ]$

$v \in \{\cancel{3}\}$

$key = [1, 0, 2, 1, 1]$

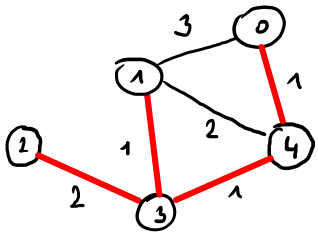
$\pi = [4, NULL, 3, 1, 3]$



Obrázek 2.10: Příklad, část 3.

6.) $key = [1, 0, 2, 1, 1]$

$\pi = [4, NULL, 3, 1, 3]$



Obrázek 2.11: Příklad, část 4.

Kapitola 3

GAL – Hledání nejkratších cest ze zdrojového uzlu do všech ostatních uzlů grafu (Bellman-Fordův algoritmus, Dijkstrův algoritmus).

3.1 Metadata

- Předmět: Grafové algoritmy (GAL)
- Přednáška:
  - 7) Nejkratší cesty z jednoho vrcholu, Bellman-Fordův algoritmus, nejkratší cesta z jednoho vrcholu v orientovaných acyklických grafech.
  - 8) Dijkstrův algoritmus. Nejkratší cesty ze všech vrcholů.
- Záznam:
  - 2020-11-05

3.2 Úvod a kontext

Viz. „Úvod a kontext“ v předchozích otázkách z tohoto předmětu.

Cena cesty Nechť $G = (V, E)$ je ohodnocený graf s váhovou funkcí $w : E \mapsto \mathbb{R}$. Cena cesty $p = \langle v_o, v_1, \ldots, v_k \rangle$ je suma

$$w(p) = \sum_{i=0}^{k} w(v_i, v_{i+1})$$

.

Cena nejkratší cesty Cena nejkratší cesty z u do v je

$$\delta(u, v) = \begin{cases} min(\{w(p) : u \overset{\text{p}}{\Rightarrow} v\}) \\ \infty \text{ pokud cesta neexistuje} \end{cases}$$

.

Nejkratší cesta Nejkratší cesta z u do v je pak libovolná cesta p taková, že $w(p) = \delta(u, v)$.

Cena cesty se záporným cyklem Pokud na cestě z u do v existuje záporný cyklus (cyklus jehož celková cena je záporná), pak $\delta(u, v) = -\infty$.

Záporné ohodnocení hran Pokud na cestě z u do v neexistuje záporný cyklus, tak algoritmy pracují dobře i se záporným ohodnocením hran.

Reprezentace cesty Cestu reprezentujeme pomocí pole předchůdců π.

Hledání nejkratších cest ze všech uzlů do jednoho Tento problém lze řešit stejnými algoritmy. Graf se transponuje (převrácení orientace hran), provede se algoritmus pro problém „hledání nejkratších cest ze jednoho uzlu do všech ostatních uzlů“ a poté se transponuje zpět.

Reprezentace nejkratší cesty Nejkratší cestu grafu $G = (V, E)$ reprezentujeme pomocí pole předchůdců π, kde $\pi[v]$ označuje předchůdce uzlu $v \in V$ na nejkratší cestě. Podgraf předchůdců pak je $G_\pi = (V_\pi, E_\pi)$, $V_\pi = \{v \in V : \pi[v] \neq \text{NULL}\} \cup \{s\}$, $E_\pi = \{(\pi[v], v) \in E : v \in V_\pi - \{s\}$. V okamžiku dokončení algoritmu výpočtu nejkratších cest je G_π strom nejkratších cest. Tj. kořenový strom obsahující nejkratší cesty ze zdroje s do všech ostatních uzlů.

3.3 Pomocné funkce

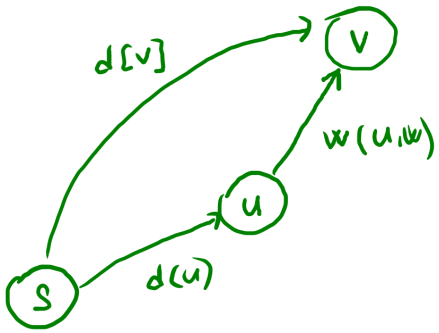
Představené algoritmy pracují z důvodu efektivity se sledy a nikoliv s cestami (bylo by nutné stále kontrolovat, zda nebyla porušena podmínka cesty), ačkoliv je problém nazývá hledání nejkratší cesty.

```
1 def initialize_single_source(G, s):
2     # G je graf
3     # s~je vychozi uzel
4     for v~in G.V:
5         d[v] = INF # d je pole vzdalenosti
6         pi[v] = NULL # pi je pole predchudcu
7     d[s] = 0
```

Výpis 3.1: Pomocná inicializační funkce. Složitost je $\Theta(n)$, kde n je počet uzlů.

```
1 def relax(u, v, w):
2     # u~a v~jsou uzly grafu
3     # w je vahova funkce
4     if d[v] > d[u] + w(u, v):
5         d[v] = d[u] + w(u, v)
6         pi[v] =
7 u~
```

Výpis 3.2: Pomocná funkce *relax*. Složitost je $O(1)$.



Obrázek 3.1: Ukázka činnosti funkce *relax*.

3.4 Bellman-Fordův algoritmus

Slouží pro řešení v obecných grafech, mohou obsahovat cykly a záporné hrany. Záporné cykly je však nutné detekovat a vrátit specifickou hodnotu. V podstatě se jedná o *brute force* algoritmus, provede se relaxace $n-1$-krát pro každou hranu.

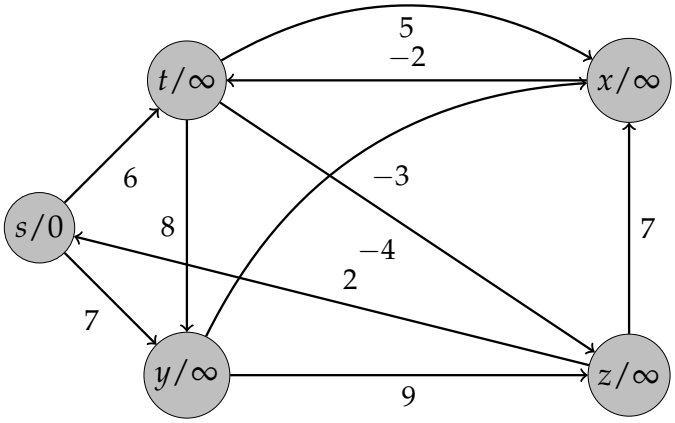
```
1  def bellman_ford(G, s, w):
2      # G je graf
3      # s~je vychozi uzel
4      # w je vahova funkce
5
6      # faze inicializace
7      initialize_single_source(G, s)
8      n = len(G.V) # pocet uzlu
9
10     # faze relaxace: provedeni (n-1) * m relaxaci (m je pocet hran)
11     for _ in range(0, n-1):
12         for u, v~in G.E:
13             relax(u, v, w)
14
15     # faze detekce zaporneho cyklu
16     for u, v~in G.E:
17         if d[u] > d[v] + w(u, v):
18             return NULL
19
20     return pi
```

Výpis 3.3: Algoritmus Bellman-Ford. Proč $n-1$ iterací? Protože mezi libovolnými dvěma uzly v grafu, existuje cesta o maximálním počtu hran $n-1$.

3.4.1 Složitost

- Řádek 7, 8 – $\Theta(1)$.
- Řádky 11, 12, 13 – $(n-1) \cdot \Theta(m) = \Theta(n \cdot m)$, kde n je počet uzlů a m je počet hran grafu.
- Řádek 16, 17, 18 – $\Theta(m)$.
- Celkem $\Theta(n \cdot m)$.

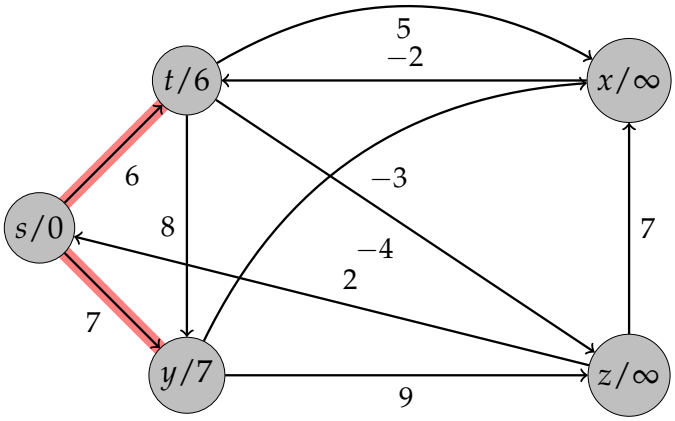
3.4.2 Příklad



Obrázek: Práce algoritmu Bellman-Ford.

- Pokud $(u,v) \in E$ je označená, pak $\pi[v] = u$
- Hrany se relaxují v tomto pořadí:
  $(t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)$.

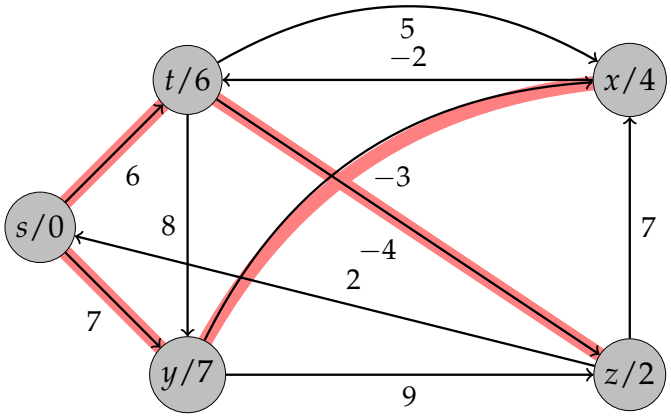
Obrázek 3.2: Příklad, část 1.



Obrázek: Práce algoritmu Bellman-Ford.

- Pokud $(u,v) \in E$ je označená, pak $\pi[v] = u$
- Hrany se relaxují v tomto pořadí:
  $(t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)$.

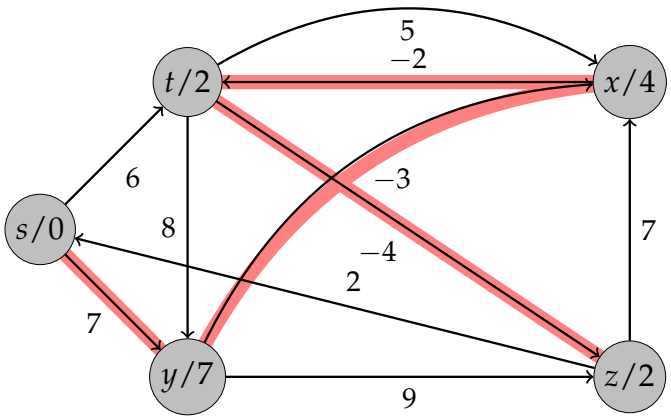
Obrázek 3.3: Příklad, část 2.



Obrázek: Práce algoritmu Bellman-Ford.

- Pokud $(u,v) \in E$ je označená, pak $\pi[v] = u$
- Hrany se relaxují v tomto pořadí:
  $(t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)$.

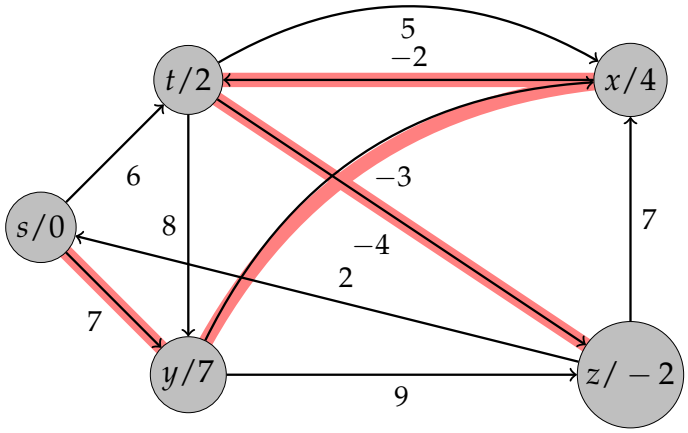
Obrázek 3.4: Příklad, část 3.



Obrázek: Práce algoritmu Bellman-Ford.

- Pokud $(u,v) \in E$ je označená, pak $\pi[v] = u$
- Hrany se relaxují v tomto pořadí:
  $(t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)$.

Obrázek 3.5: Příklad, část 4.



Obrázek: Práce algoritmu Bellman-Ford.

- Pokud $(u,v) \in E$ je označená, pak $\pi[v] = u$
- Hrany se relaxují v tomto pořadí:
  $(t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)$.

Obrázek 3.6: Příklad, část 5.

3.5 Dijkstrův algoritmus

Slouží pro řešení v acyklických grafech bez záporných hran. Pro takto omezený problém existují rychlejší algoritmy než pro problém v obecných grafech.

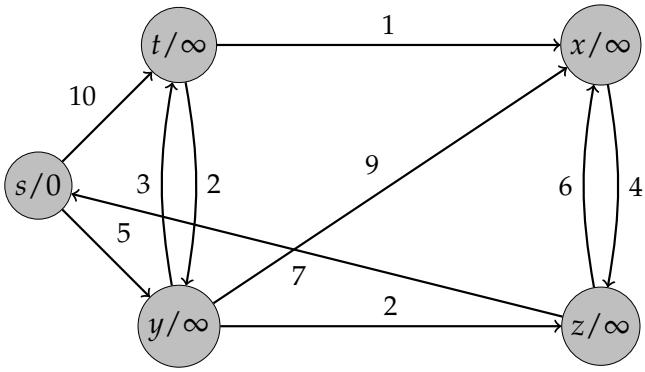
```
def dijkstra(G, s, w):
    # G je graf
    # s~je vychozi uzel
    # w je vahova funkce

    # faze inicializace
    initialize_single_source(G, s)
    Q = Queue(G.V) # prioritni fronta uzlu
    S~= {}# mnozina uzlu, ktera uz byla prozkoumana

    # faze relaxace
    while not Q.empty():
        u~= Q.extract_min(d) # vrati prvek z~Q s~nejmensi hodnotou v~d
        S~+= {u}
        # pro vsechny sousedy uzlu u~(Adj je seznam sousedu)
        for v~in Adj[u]:
            relax(u, v, w)

        Q.decrease_key(d) # aktualizace prioritni fronty

    return d, pi
```

Výpis 3.4: Algoritmus Dijkstra.

3.5.1 Složitost

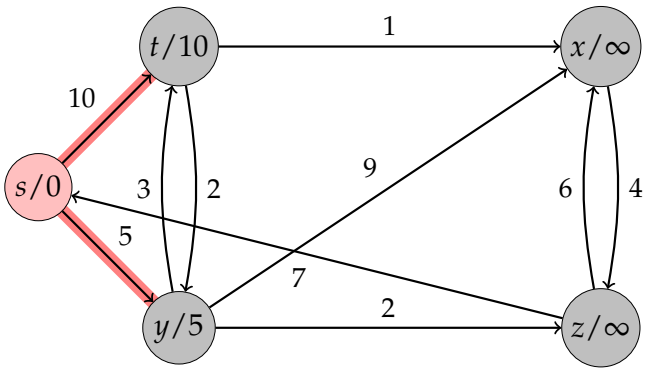
- Předpokládejme implementaci prioritní fronty pomocí pole.
- Řádek 8, 18 – $O(1)$.
- Řádek 11 – While cyklus se provede n-krát, kde n je počet uzlů.
- Řádek 12 – $O(n)$, najítí minima v poli uzlů. Celkově (s cyklem) $O(n^2)$.
- Řádek 16 – $O(m)$, pro všechny hrany. Celkově (s cyklem) $O(m \cdot n)$.
- Celkem $O(n^2 + m) = O(n^2)$.
- Pro řídké grafy lze využít implementaci fronty pomocí binární haldy a získat tak $O(m \cdot \log(n))$.
- Při implementaci fronty pomocí Fibonacciho haldy dostaneme časovou složitost $O(n \cdot \log(n) + m)$.

3.5.2 Příklad



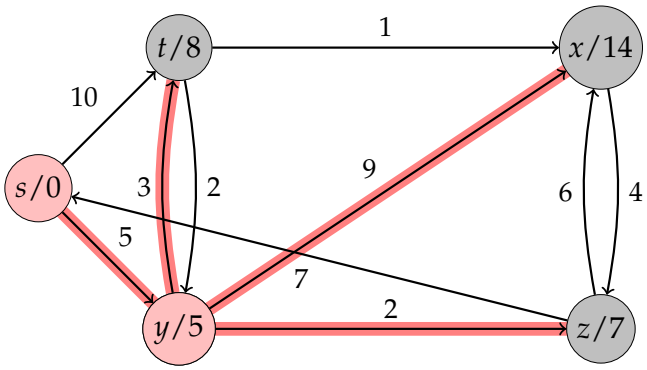
Obrázek: Práce Dijkstrova algoritmu. Označené uzly značí uzly z množiny S.

Obrázek 3.7: Příklad, část 1.



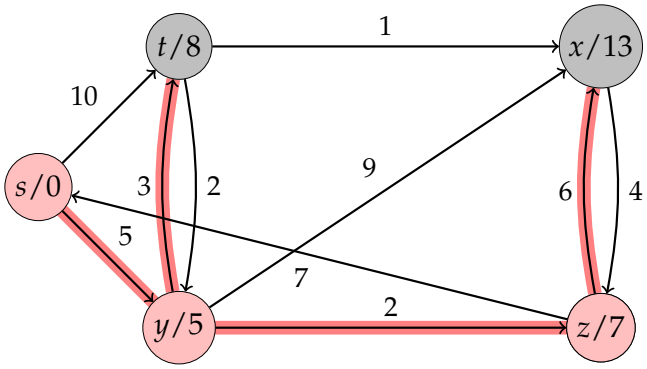
Obrázek: Práce Dijkstrova algoritmu. Označené uzly značí uzly z množiny S.

Obrázek 3.8: Příklad, část 2.



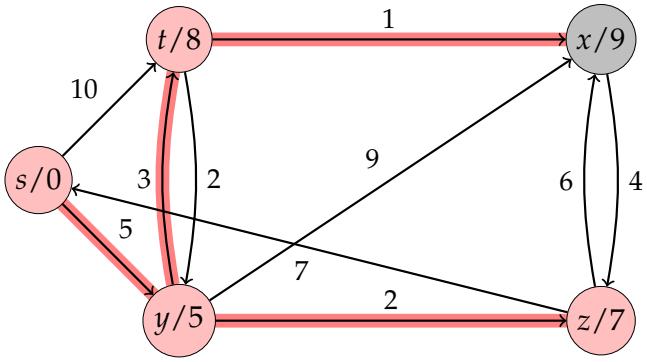
Obrázek: Práce Dijkstrova algoritmu. Označené uzly značí uzly z množiny S.

Obrázek 3.9: Příklad, část 3.



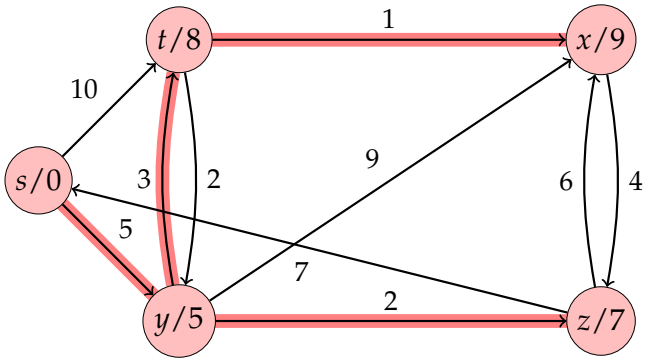
Obrázek: Práce Dijkstrova algoritmu. Označené uzly značí uzly z množiny S.

Obrázek 3.10: Příklad, část 4.



Obrázek: Práce Dijkstrova algoritmu. Označené uzly značí uzly z množiny S.

Obrázek 3.11: Příklad, část 5.



Obrázek: Práce Dijkstrova algoritmu. Označené uzly značí uzly z množiny S.

Obrázek 3.12: Příklad, část 6.

Kapitola 4

PDI – Klasifikace algoritmů volby koordinátora, algoritmus Bully a jeho složitost.

4.1 Metadata

- Předmět: Prostředí distribuovaných aplikací (PDI)
- Přednáška:
  - 7) Synchronizace
- Záznam:
  - 2020-11-02

4.2 Úvod a kontext

- Mějme množinu procesů v rámci distribuovaného systému. Řešíme problém nalezení shody na nějaké věci (synchronizační problém). Problém můžeme rozdělit na dvě situace:
  - **Problém volby koordinátora** – Výběr jednoho z procesů, který bude vedoucím procesem (koordinátor). Tento proces pak může vykonat určitou činnost nebo může sloužit ostatním procesům k realizaci význačné role v systému.
  - **Problém vzájemného vyloučení** – Předpokládejme, že konkrétní zdroj může v daném okamžiku používat pouze jeden proces. Tento problém se běžně vyskytuje ve víceprocesorových systémech, ale také v distribuovaných systémech.
- Synchronizační problémy lze v rámci operačních systémů nebo multiprocesorových systémů řešit pomocí provádění atomických operací, sdílené paměti apod. (je pro ně podpora v rámci operačního systému nebo hardwaru). V distribuovaných systémech nic takového není z principu možné a proto se synchronizační problémy řeší pomocí zasílání zprav, resp. algoritmicky.

4.3 Problém volby koordinátora

- Předpokládáme:

  - Každý proces má unikátní ID.
  - Procesy neznají stav (běžící, neběžící) dalších procesů.
  - Každý proces zná ID dalších procesů (záleží na topologii).
- Cíl:
  - Dosáhnutí shody mezi všemi procesy na procesu, který je koordinátor.
  - Kritérium výběru koordinátora může být různé. Např. na základě proces ID (proces s největším ID se stane koordinátorem).

4.4 Bully algoritmus

Pro topologii každý s každým – každý proces může komunikovat s každým dalším procesem. Používá tři druhy zpráv: ELECTION, OK, COORDINATOR.

4.4.1 Postup

- Proces P, který má podezření, že chybí koordinátor, může zahájit volby.
  1. Proces P odešle zprávu ELECTION všem procesům s větším ID.
  2. Pokud nikdo neodpoví, P vyhrává volby a stává se koordinátorem.
  3. Pokud některý z procesů s větším ID odpoví (zpráva OK), tak přebírá řízení a práce P je ukončena.
  4. Pokud P obdrží zprávu ELECTION od procesů s menším ID, pošle jim odpověď OK na zablokování procesů.
- Nakonec zůstane pouze P (nový koordinátor), který o tom informuje ostatní zasláním zprávy COORDINATOR.
- Pokud se proces probudí nebo je restartován, první akcí je vyvolání voleb.

4.4.2 Složitost

Složitost z hlediska počtu zpráv.

Nejhorší případ (iniciátor s nejmenším ID):

- $(n-1)$ iterací
- $2(n-1)$ zpráv ELECTION a OK pro každou iteraci
- $(n-1)$ zpráv COORDINATOR
- Celkem: $(n-1) \times 2(n-1) + (n-1) \approx n^2$

Nejlepší případ (iniciátor s největším ID):

- $(n-1)$ zpráv COORDINATOR
- Celkem: $(n-1)$

4.4.3 Příklad

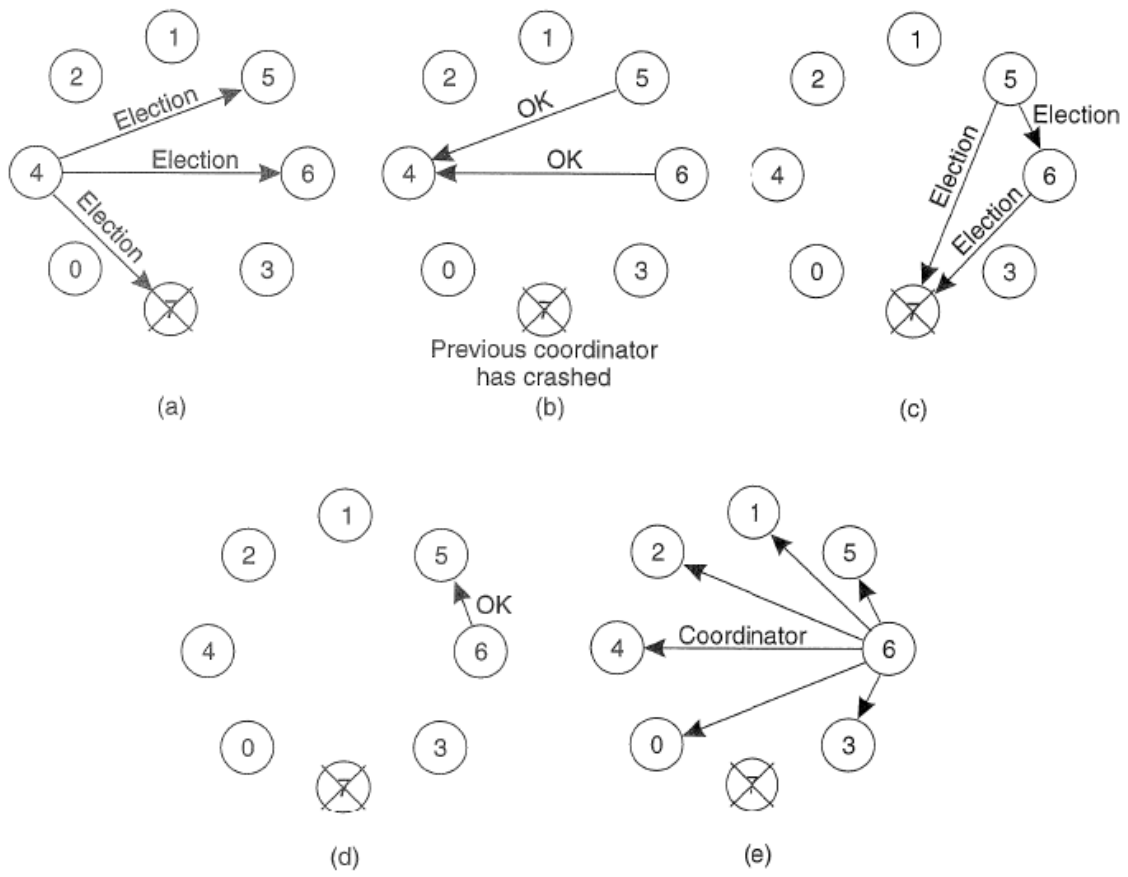


Figure 5-11. The bully election algorithm. (a) Process 4 holds an election. (b) Processes 5 and 6 respond, telling 4 to stop. (c) Now 5 and 6 each hold an election. (d) Process 6 tells 5 to stop. (e) Process 6 wins and tells everyone.

Obrázek 4.1: Příklad činnosti Bully algoritmu.

4.5 Ring Algoritmus

Pro kruhovou topologii – procesy jsou uspořádané do kruhu podle svého proces ID. Každý proces musí vědět nejenom o svém následovníkovi, ale také o jeho následníkovi, který funguje jako „záloha“, v případě že by se přímý následník stal nedostupný. Používá dva druhy zpráv: ELECTION, COORDINATOR.

4.5.1 Postup

- Proces P, který má podezření, že chybí koordinátor, může zahájit volby.
  1. Zašle zprávu ELECTION obsahující jeho ID dalšímu procesu (pokud další proces nereaguje, proces P zašle stejnou zprávu dalšímu v kruhu).
  2. Každý člen topologie přijme zprávu ELECTION, přidá do ní své ID a přepošle zprávu dalšímu procesu.

- Když se zpráva vrátí k procesu P, je zpráva převedena na zprávu COORDINATOR a poslána následujícímu procesu v topologii, aby bylo možné nahlásit:
  1. Novým koordinátorem se stává proces s nejvyšším ID.
  2. Členové sítě jsou stále aktivní.
- Po síti může obíhat více zpráv zároveň.

4.5.2 Složitost

Složitost z hlediska počtu zpráv.

Vždy $2n \approx n$ zpráv. Jedno kolečko „oběhne“ zpráva ELECTION a druhé zpráva COORDINATOR.

4.5.3 Příklad

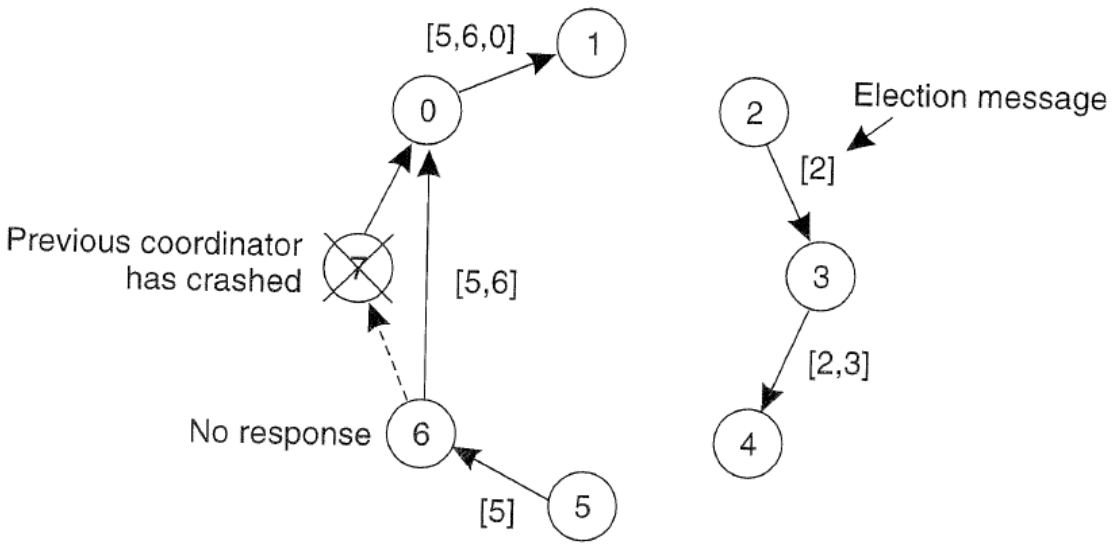


Figure 5-12. Election algorithm using a ring.

Obrázek 4.2: Příklad činnosti Ring algoritmu.

4.6 Algoritmus pro obecnou topologii

Předpokládáme, že nemáme ani kruhovou topologii ani spojení každý s každým. Např.: peer-to-peer sítě, sensorové sítě, ...

4.6.1 Postup

- V první iteraci se broadcastem posílá zpráva ELECTION.
- Každý uzel si uloží od kterého souseda dostal zprávu ELECTION jako první. Tím vzniká kostra grafu (*spanning tree*).

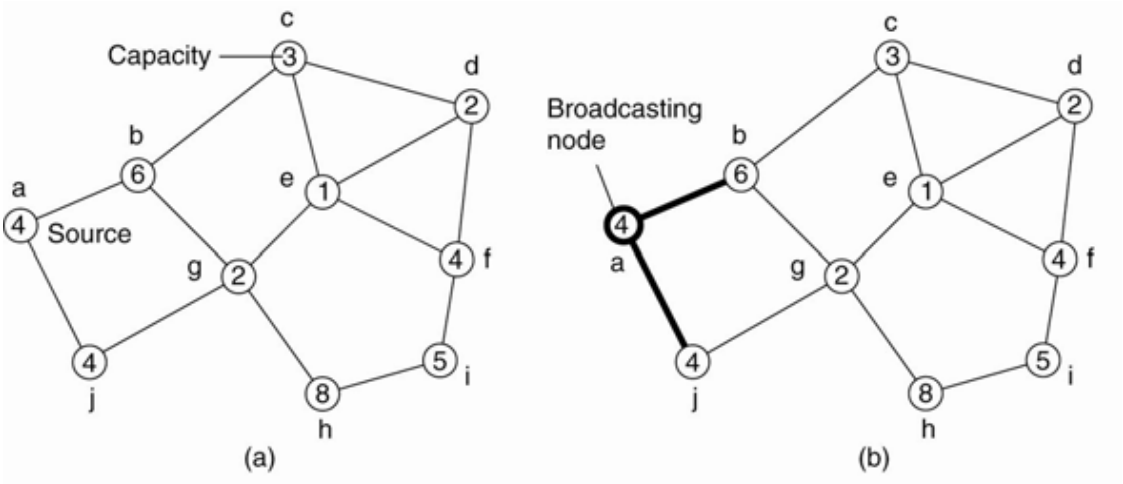
- Uložený soused je poté využijí pro zpětnou komunikaci. To znamená, že další komunikace už probíhá přes strom, nikoliv přes broadcast. Tím je ušetřena některé komunikace.

4.6.2 Složitost

Složitost z hlediska počtu zpráv.

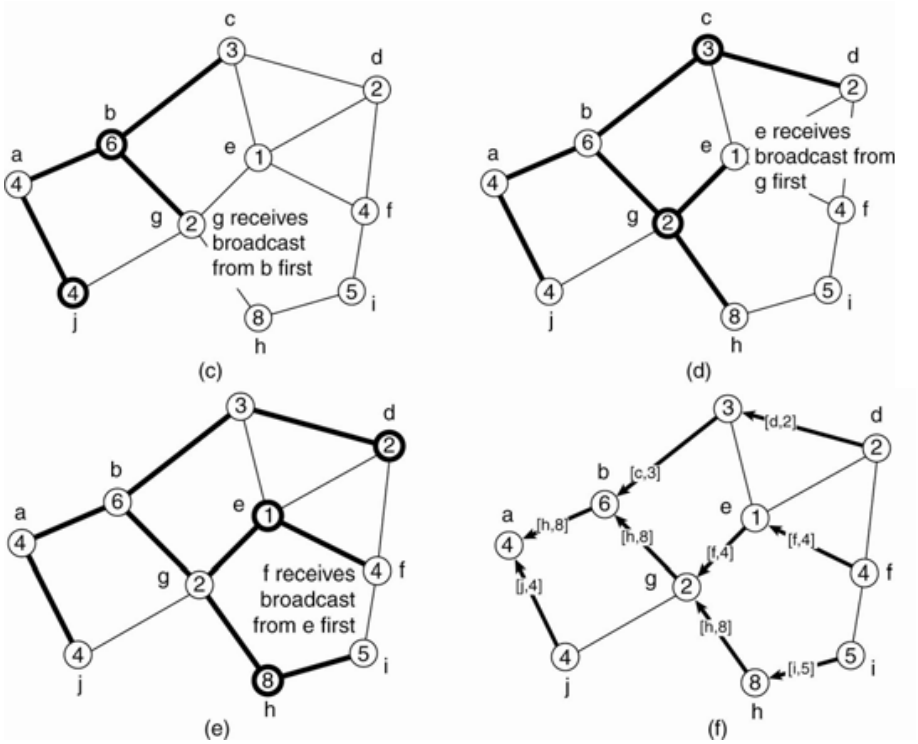
- Inicializační broadcast: počet hran grafu.
- Odpověď: počet hran kostry grafu.
- Result broadcast: počet hran kostry grafu.

4.6.3 Příklad



Node *a* initiates an election.

Obrázek 4.3: Příklad činnosti algoritmu pro obecnou topologii, část 1.



In the end, source *a* notes that *h* is the best leader and broadcasts this info to all nodes.

Obrázek 4.4: Příklad činnosti algoritmu pro obecnou topologii, část 2.

Kapitola 5

PDI – Podmínky konsistentního globálního stavu distribuovaného systému.

5.0.1 Metadata

- Předmět: Prostředí distribuovaných aplikací (PDI)
- Přednáška:
  - 4) Globální stav a snapshots
- Záznam:
  - 2020-10-12

5.1 Úvod a kontext

Distribuovaný systém Distribovaný systém je množina procesů $p_1, p_2, \ldots, p_n$, které jsou propojeny komunikačními kanály. V systému neexistuje žádná globální paměť ani globální hodiny. Procesy spolu komunikují pouze zasíláním zpráv skrze komunikačními kanály.

Komunikační kanál Komunikační kanál mezi procesy p_i a p_j značíme C_{ij}.

Událost Rozlišujeme tři typy událostí: interní událost procesu, zaslání zprávy a přijetí zprávy.

Zpráva Zpráva m_{ij} značí zprávu zaslanou procesem p_i procesu p_j. $send(m_{ij})$ značí odeslání zprávy a $recv(m_{ij})$ přijetí.

Stav procesu Lokální stav procesu p_i značíme LS_i. Lokální stav je definován jako sekvence všech událostí, o kterých proces p_i ví. Nechť e je libovolná událost, $e \in LS_i$ značí, že událost e patří do lokálního stavu procesu p_i, $e \notin LS_i$ značí, že událost e nepatří do lokálního stavu procesu p_i.

Stav komunikačního kanálu Stav komunikačního kanálu C_{ij} značíme SC_{ij} a je definován množinou zpráv, které obsahuje. Pro kanál C_{ij} můžeme definovat jeho stav na základě

lokálních stavů procesů LS_i a LS_j:

$$transit(LS_i, LS_j) = \{m_{ij} \mid send(m_{ij}) \in LS_i \wedge rec(m_{ij}) \notin LS_j\}$$

.

5.2 Model komunikace

- FIFO – Komukační kanál funguje jako fronta zpráv *first in*, *first out*. Kanál tedy zachovává pořadí zpráv sám o sobě.
- non-FIFO – Komunikační kanál se chová jako datová struktura množina, do které odesílatel vkládá zprávy a příjemce je odebírá v náhodném pořadí.
- Causal ordering (kauzální uspořádání) – Systém, který podporuje kauzální doručení zpráv splňuje následující vlastnost. Pro jakékoliv dvě zprávy m_{ij} a m_{kj} platí, pokud $send(m_{ij}) \rightarrow send(m_{kj})$, pak i $recv(m_{ij}) \rightarrow recv(m_{kj})$.

5.3 Konzistentní globální stav

Globální stav Globální stav distribuovaného systému je kolekce lokálních stavů procesů a komunikačních kanálů.

$$GS = \left\{ \bigcup_i LS_i \,, \bigcup_{i,j} SC_{ij} \right\}$$

.

Časoprostorový diagram Diagram pro vizualizaci komunikace procesů v distribuovaném systému. Viz obrázek 5.1 a 5.2.

Konzistentní globální stav Konzistentní globální stav (*snapshot*) je stav systému v určitém časovém okamžiku. Lze si jej představit jako řez v časoprostorovém diagramu, který rozděluje diagram na dvě části: minulost a budoucnost. Aby byl řez (globální stav) konzistentní, tak pokud je doručení nějaké zprávy v minulosti, musí být v minulosti i její odeslání. Formálně jde o globální stav, který splňuje nálsedující podmínky:

$$send(m_{ij}) \in LS_i \Rightarrow m_{ij} \in SC_{ij} \oplus recv(m_{ij}) \in LS_j$$

,

$$send(m_{ij}) \notin LS_i \Rightarrow m_{ij} \notin SC_{ij} \wedge recv(m_{ij}) \notin LS_j$$

.

K čemu je *snapshot* *Snapshot* lze využít např. pro tvorbu záloh systému nebo při zotavování systému po chybách.

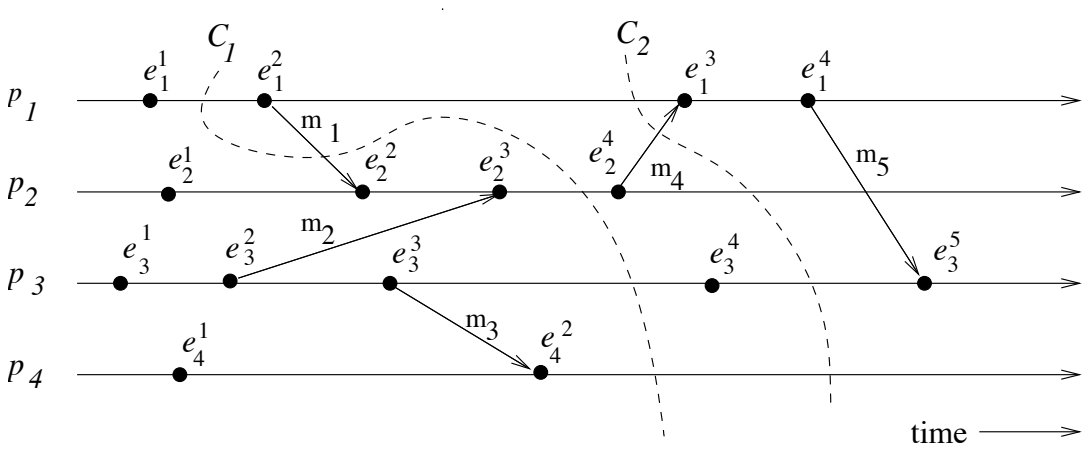
Jak lze *snapshot* vytvořit Absence globální sdílené paměti, globálních hodin a nepředvídatelná délka zpoždění v odesílání zpráv v distribuovaném systému činí problém vytváření snapshotů netriviálním. Způsob vytváření lze rozdělit do dvou kategorií: na základě algoritmů a na základě checkpointů.

Problémy při zaznamenávání snapshotu Jak rozlišit mezi zprávami, které mají být součástí snapshotu a které nikoliv?

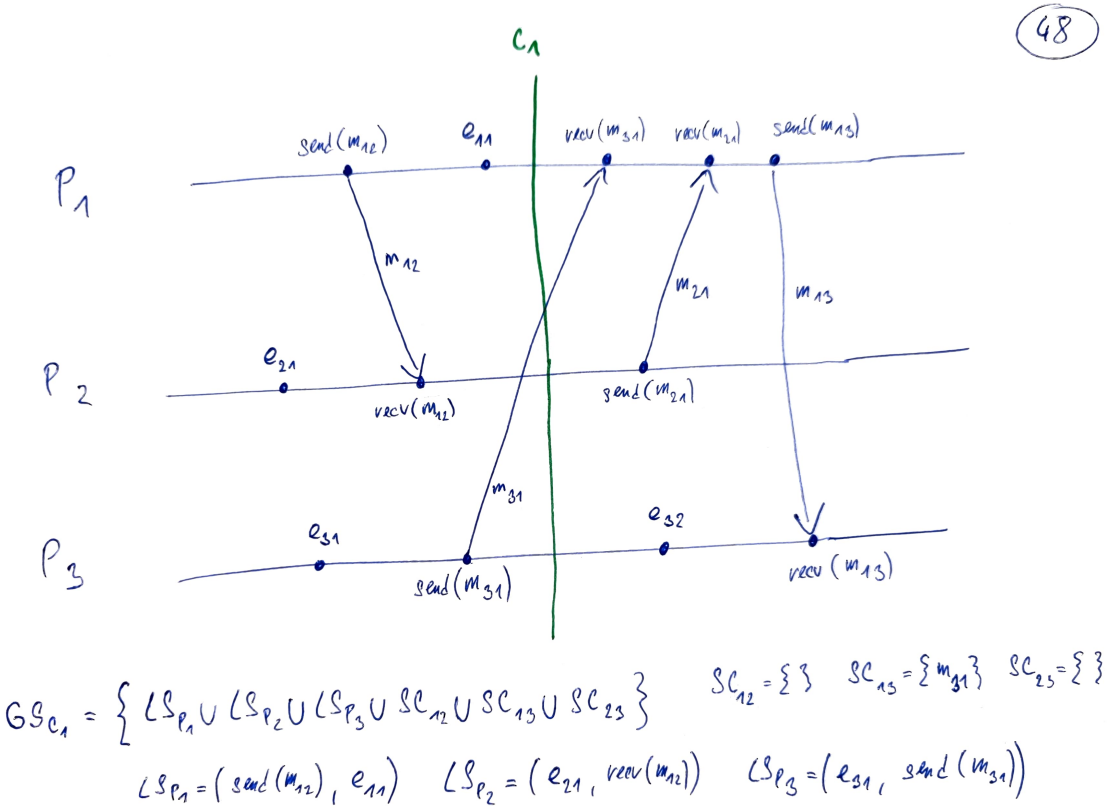
- Zprávy, které jsou odeslány procesem před zaznamenáním svého snapshotu, jsou zaznamenány do stavu.
- Zprávy, které jsouodeslány procesem po zaznamenání svého snapshotu, nejsou zaznamenány do stavu.

Jak rozpoznat okamžik, ve kterém má proces zaznamenat snapshot?

- Proces p_j musí zaznamenat svůj snapshot před zpracováním zprávy m_{ij}, která byla poslána procesem p_i po zaznamenání jeho snapshotu.



Obrázek 5.1: Příklad řezu v časoprostorovém diagramu. Řez C_1 je nekonzistentní, kvůli zprávě m_1. Řez C_2 je konzistentní a zpráva m_4 je zachycena ve stavu kanálu Ch_{21}.



Obrázek 5.2: Příklad konzistentního globální stavu formálně.

Kapitola 6

PDI – Principy distribuovaného zpracování MapReduce, průběh a jednotlivé operace distribuovaného výpočtu pomocí MapReduce, jeho implementace v Apache Hadoop a Apache Spark.

6.1 Metadata

- Předmět: Prostředí distribuovaných aplikací (PDI)
- Přednáška:
  - 9) Programovací model MapReduce a Apache Hadoop
  - 10) Distribuované souborové systémy
  - 11) Apache Spark
- Záznam:
  - 2020-11-16
  - 2020-11-23

6.2 Úvod a kontext

OLTP OLTP (*Online Transactional Processing*, provozní databáze, systémy pro online zpracování transakcí) jsou standardní databázové systémy s pevnou strukturou dat definovou pomocí databázového schématu. Jsou navrženy a optimalizovány pro chod provozních aplikací s primárním cílem zajistit rychlý a souběžný přístup k datům. To vyžaduje transakční zpracování, řízení souběžnosti a techniky obnovy (rollback), které zaručují konzistenci dat. Díky těmto vlastnostem mají OLTP databáze špatný výkon při provádění složitých dotazů, které potřebují spojit mnoho relačních tabulek dohromady nebo agregovat velké objemy dat. Kromě toho obsahují typicky podrobná data a neobsahují historická data, která jsou při datové analýze potřeba.

OLAP OLAP (*Online Analytical Processing*, online analytické zpracování) je databázové paradigma specificky zaměřené na dotazy, zejména na analytické dotazy. Používají se zde jiné techniky indexování a optimalizace dotazů. Normalizace není pro toto paradigma žádoucí, protože rozděluje databázi na mnoho tabulek. Složité dotazy v takovém případě vyžadují rekonstrukci dat a s tím spojený vysoký počet spojování tabulek. Pracuje se s tzv. multidimenzionálními kostkami, avšak v pozadí jsou stále relační databáze.

NoSQL Potřeba ukládat proudy dat (zpracovávané v reálném čase bez možnosti pozastavení), obrázky, multimédia, velké JSON soubory, ..., vedla ke vzniku NoSQL databází. NoSQL databáze používají jiné prostředky než tabulková schémata tradiční relační databáze. Často jde o „hloupé“, nestrukturované uložiště klíč-hodnota.

BigData Velká, nestrukturovaná (různorodá), rychle rostoucí data, která není možné uložit ani zpracovávat běžnými přístupy (na jednom uzlu, jedním uzlem). Produkují je např.: IoT senzory, sociální sítě, chatovací aplikace, webové vyhledávače, ... Pro jejich zpracování je nutné využít distribuované systémy (pro uložení i zpracování).

Distribuované zpracování dat Distribuované zpracování dat je zpracování velkých dat (*big data*) pomocí distribuovaných systémů. To s sebou přináší problémy. Jak zaručit vhodnou distribuci dat a výpočtu mezi uzly? Jak řešit nespolehlivost a výpadky uzlů? Jak a kam zajistit doručení výsledků výpočtu? ...

6.3 MapReduce

Algoritmy pro indexování webových stránek (Page Rank) přestávaly být udržitelné, bylo potřeba zvýšit jejich škálovatelnost. Google vydal příspěvek „MapReduce: Simplified Data Processing on Large Clusters“, kde bylo představeno paradigma MapReduce. Jde o paradigma distribuovaného výpočtu založené na funkcích *map* a *reduce* z funcionálního programování.

Map Funkce *map* má ve funkcionálním programování 2 vstupní parametry a vrací seznam hodnot. První parametr je unární operátor (nebo funkce fungující jako unární operátor) a druhý je seznam hodnot. Výstupní seznam je spočítán jako aplikace unárního operátoru na vstupní seznam. Příklad:

$$map(square, [1, 2, 3, 4]) = [1, 4, 9, 16]$$

. V paradigmata MapReduce *map* vrací data jako seznam dvojic klíč-hodnota, přesněji:

$$map((key, value)) \rightarrow [(key, value)]$$

.

Reduce Funkce *reduce* má ve funkcionálním programování 2 vstupní parametry a vrací jednu hodnotu. První parametr je binární operátor (nebo funkce fungující jako binární operátor) a druhý je seznam hodnot. Výstupní hodnota je spočítána jako postupná aplikace binárního operátoru na všechny hodnoty ve vstupním seznamu. Příklad:

$$reduce(+, [1, 4, 9, 16]) = 30$$

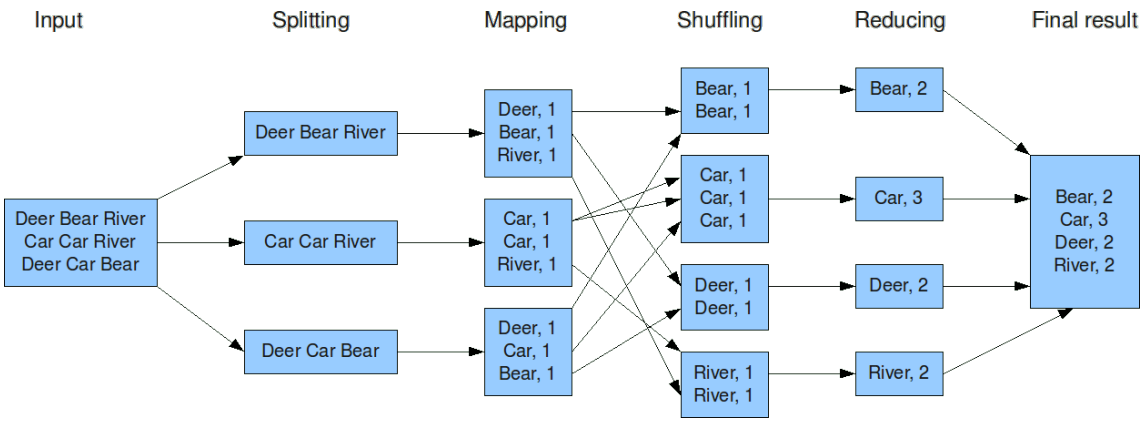
. V paradigmata MapReduce *reduce* bere na vstupu klíč a seznam hodnot a vrací opět seznam dvojic klíč-hodnota, přesněji:

$$reduce(key, [value]) \rightarrow [(key, value)]$$

.

```python
def map(input_key: str, input_value: str) -> list[tuple[str, int]]:
    # input_key - document name
    # input_value - document content (etc. line)
    result = []
    for word in input_value.split(' '):
        result.append((word, 1))
    return result

def reduce(input_key: str, input_value: list[int]) -> tuple[str, int]:
    result = 0
    for val in input_value:
        result += value
    return (input_key, result)
```

Výpis 6.1: Příklad implementace funkcí *map* a *reduce* v paradigmatu MapReduce pro počítání četnosti slov ve vstupu v Pythonu.

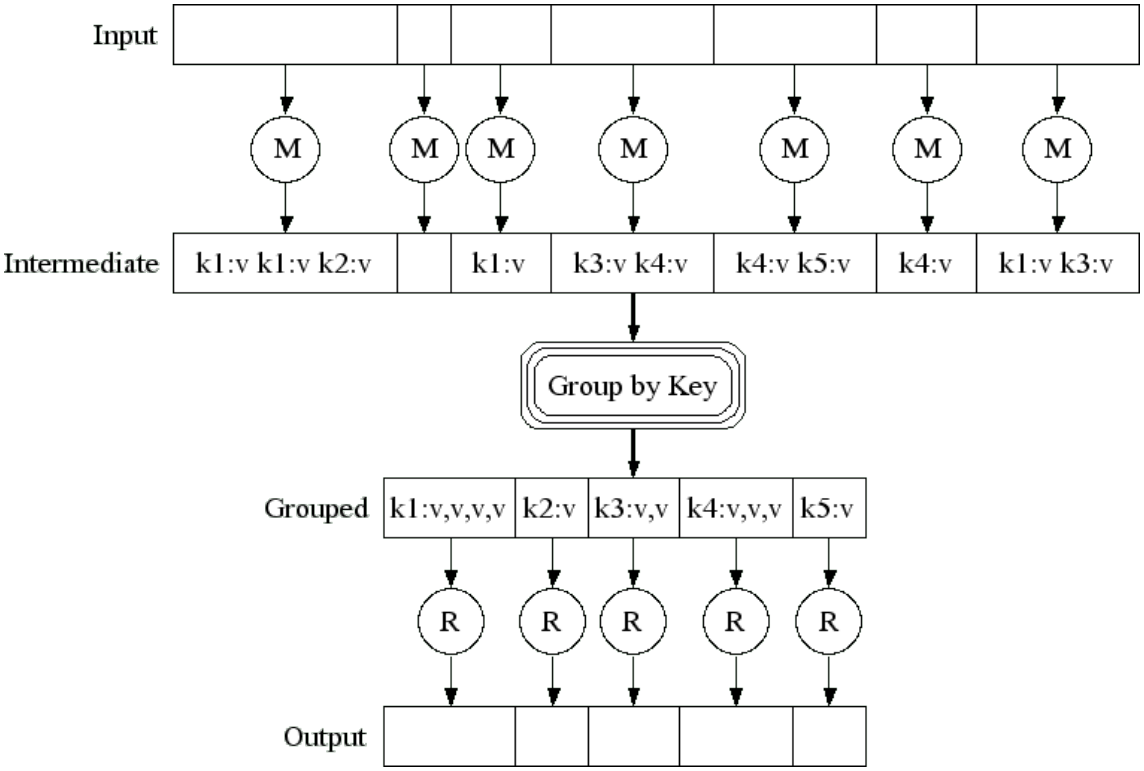


Obrázek 6.1: Úloha počítání četnosti slov v paradigmatu MapReduce v diagramu.

Průběh MapReduce Celý MapReduce probíhá v několika krocích, viz obrázek 6.1.

1. Input – Přípravený vstup pro distribuovaný výpočet (např. soubory ve virtuálním distribuovaném souborovém systému, viz dále HDFS).
2. Splitting – Rozdělení vstupu na části, které budou přiděleny jednotlivým uzlům. Může být výchozí (např. rozdělení textového souboru po řádcích) nebo definováno uživatelem.
3. Mapping – Každý uzel aplikuje funkci *map* na svoji přidělenou část. Uživatel definuje jak má funkce *map* vypadat.
4. Shuffling (také Grouping, Partitioning, Comparing)– Výpočetní uzly si mezi sebou vymění hodnoty, které spočítaly, na základě klíče. Tento krok zařižuje platforma pro distribuovaný výpočet sama o sobě, typicky na základě hashů klíčů. Tento krok je většinou *bottleneck*.

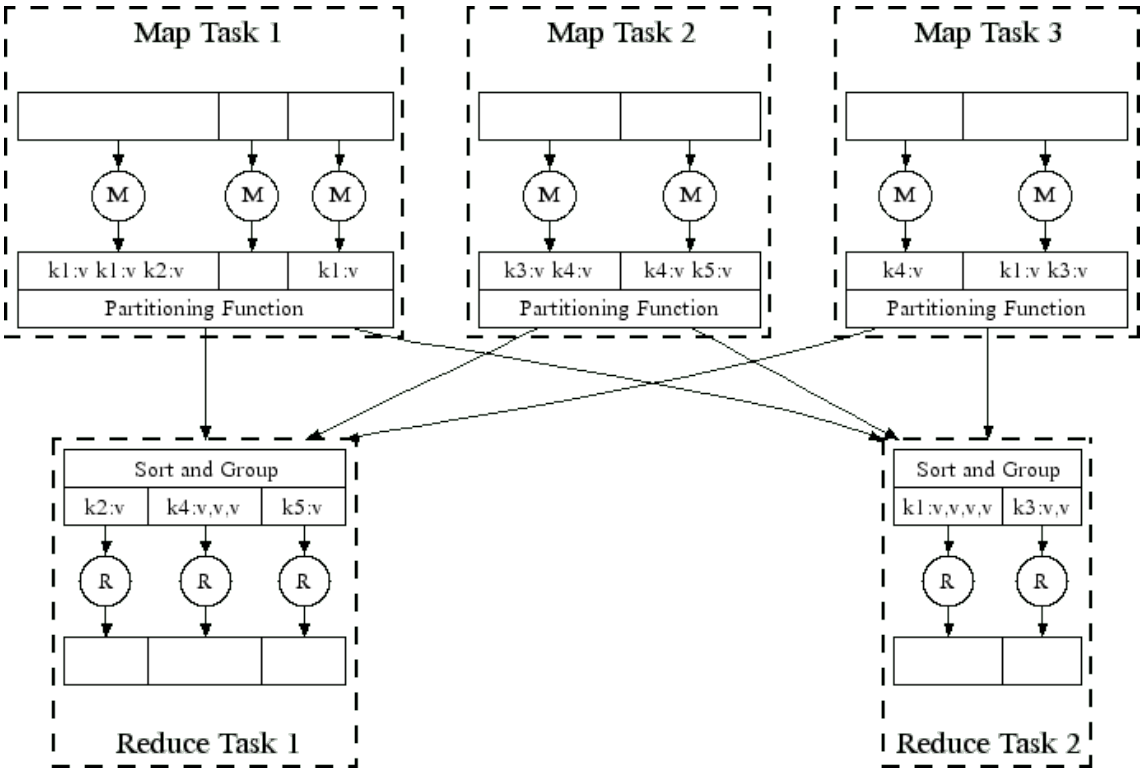
5. Reducing – Každý uzel zapojený do tohoto kroku (často je v tomto kroku potřeba méně uzlů, než v kroku mapping) aplikuje funkci *reduce* na svoji přidělenou část. Uživatel definuje jak má funkce *reduce* vypadat.
6. Final Result – Finální výsledek (např. zapsán do do virtuálního distribuovaného souborového systému, viz dále HDFS).



Obrázek 6.2: Výpočet MapReduce v obecném schématu.

Combiner Optimalizační krok, jde o „jakési“ provedení operace *reduce* už ve fázi *map* (každým uzlem). Tím je snížen počet mezivýsledků ve fázi Shuffling. Typicky funkce *combine* je stejná jako *reduce*.

Virtuální distribuovaný souborový systém Pro realizaci distribuovaného výpočtu je rovněž potřeba distribuovaný souborový systém (DFS). Ten je typicky realizován jako virtuální souborový systém nad jednotlivými souborovými systémy uzlů. Např.: GFS – Google File System, HDFS – Hadoop File System (viz dále). DFS obsahuje data samotná (*data nodes*) a metadata o tom, která data jsou na jakých uzlech (*name nodes*).



Obrázek 6.3: Výpočet MapReduce v obecném schématu a rozdělením práce na jednotlivé uzly (uzel je typicky víceprocesorový).

6.4 Apache Hadoop

Apache Hadoop je *open-source* implementace MapReduce paradigmatu vyvíjená Apache Software Foundation. Jde o implementaci v Jave, ta je vhodná, jelikož díky JVM (Java Virtual Machine) je spouštění uživatel definovaných funkcí *map* a *reduce* snadné.

Hadoop MapReduce – Implementace MapReduce paradigma. Data jsou čtena a ukládána na HDFS (včetně mezivýsledků). To znamená, můžeme pracovat v podstatě neomezenými daty, ale ukládání a načítání výpočet zpomalují.[1].

HDFS HDFS (*Hadoop Distribute File System*) je virtuální distribuovaný souborový systém. Standardní soubor je rozdělen na datové bloky které jsou distribuovány na různé datové uzly. Architektura HDFS se skládá ze dvou typů uzlů – Name Node a Data Node. Name Node obsahuje alokační tabulku pro souborový systém. Ví které datové bloky patří kterému souboru a kde jsou uloženy. Obsahuje další metadata jako názvy souborů, cesty, ... Data Node obsahuje datové bloky. Typicky redundance a replikace, počítá se s možným selháním uzlů. Pro **čtení dat** se klient zeptá Name Nodu na konkrétní soubor v HDFS. Name Node vrátí metadata o souboru, na jakých Data Nodech se vyskytuje. Klient požádá příslušné Data Nody, ty mu pošlou data, která se na klientovi „poskládají“ do výsledného souboru. Pro **zápis dat** se klient zeptá Name Nodu, kam by měl zapisovat. Klient zapíše na příslušný Data Node. Data Node poté vyřeší replikace s dalšími uzly.

[1] *Nebylo přednášeno podrobněji, pravděpodobně stačí princip obecného MapReduce, který byl vysvětlen v předchozí sekci.*

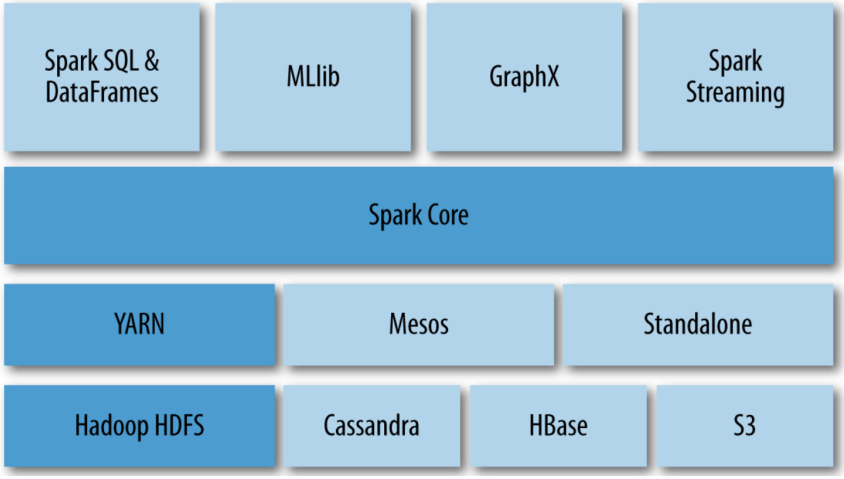
Hadoop YARN Hadoop YARN je plánovač (*scheduler*). Plánuje výpočet tak, aby proběhl co nejlepším způsobem na konkrétnbí distribuované architektuře. Plánovač má obecné obecné rozhraní a Hadoop YARN lze nahradit za jiný.

Hadoop Common Hadoop Common jsou další knihovny a ovladače pro klienty.

Další nástroje Nad Apache Hadoop existuje mnoho dalších nástrojů. Apache Pig pro *high level* programování map-reduce úloh. Apache Hive pro pro dolování dat nad Apache Hadoop. Apache HBase jako distribuovaná databáze nad Apache Hadoop, ...

6.5 Apache Spark

Apache Spark je *open-source* nástroj pro distribuované zpracování rozsáhlých dat vyvíjený Apache Software Foundation. Hlavní cíl je zvýšení rychlosti. Spark na to jde přesunutím co nejvíce výpočtů do operační paměti jednotlivých uzlů a tím pádem zminimalizovat počet zápisů a čtení z DFS (snaha odstranit *bottleneck* v kroku shuffling u Hadoopu). Tím ale vzniká jiný problém, a sice výpadek uzlu znamená, že data jsou ztraceny.



Obrázek 6.4: Architektura Apache Spark. Hlavní je Spark Core, zbytek funguje na systému pluginů a může používat HDFS, Hadoop YARN a Hadoop Common.

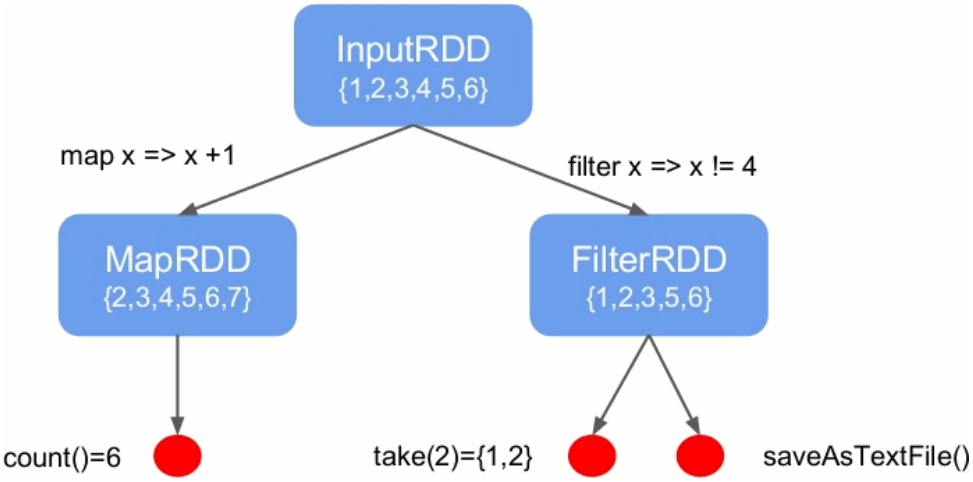
Resilient Distributed Dataset Resilient Distributed Dataset (RDD) je základní datová struktura Sparku. Jedná se o typované kolekce n-tic, které jsou neměnné (*read only*). Vstup je transformován na RDD a každá operace je pak transformace jednoho RDD na jiné.

$$RDD_1 \rightarrow map() \rightarrow RDD_2 \rightarrow reduce() \rightarrow RDD_3$$

Strategie vyhodnocování Spark uplatňuje strategii vyhodnocování *lazy evaluation*. Vyhodnocování výrazu je odkládáno až do doby, dokud není potřeba jeho hodnota. Zabraňuje opakovanému vyhodnocování. Je vyhodnocována pouze ta část, která je potřeba. RDD funguje jako abstraktní datová struktura, nemusí obsahovat data uvnitř, ale pouze předpis jak data získat a získá je, až když jsou potřeba.

Struktura výpočtu Struktura výpočtu odpovídá orientovanému acyklickému grafu (DAG, *Directed Acyclic Graph*). Uzly jsou RDD a hrany jsou transformace. DAG je znám i dalším uzlům, takže pokud nastane výpadek uzlu a výpočet je ztracen, může být uzel snadno zastoupen.

Klíčové vlastnosti Klíčové vlastnosti Sparku jsou *lazy evaluation*, *in-memmory* a *parallel computing*.



Obrázek 6.5: Příklad výpočtu v Apache Spark.

Kapitola 7

KRY – Symetrická kryptografie. Vlastnosti, vlastnosti bezpečného algoritmu, délka klíče, útok silou, příklady symetrických algoritmů, Feistelovy šifry, DES, režimy činnosti, proudové šifry.

7.1 Metadata

- Předmět: Kryptografie (KRY)
- Přednáška:
  - 3) Symetrická kryptografie. Vlastnosti, vlastnosti bezpečného algoritmu, délka klíče, útok silou.
  - 4) Příklady symetrických algoritmů, Feistelovy šifry, DES, struktura, činnost, slabiny, režimy činnosti.
  - 5) Typické aplikace symetrické kryptografie.
- Záznam:
  - 2021-02-22
  - 2021-03-01
  - 2021-03-08

7.2 Úvod a kontext

Kryptografie Kryptografie (šifrování) je věda o metodách utajování smyslu zpráv převodem do podoby, která je čitelná jen se speciální znalostí.

Kryptoanalýza Kryptoanalýza je věda zabývající se metodami získávání obsahu šifrovaných informací bez přístupu k tajným informacím, které jsou za normálních okolností potřeba, tzn. především k tajnému klíči.

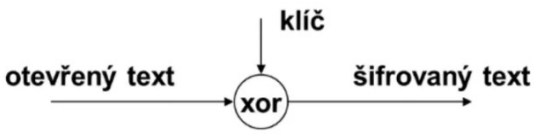
Kryptologie Jeden výraz pro kryptografii a kryptoanalýzu.

Caesarova šifra Princip Caesarovy šifry je založen na tom, že všechna písmena zprávy jsou během šifrování zaměněna za písmeno, které se abecedně nachází o pevně určený počet míst dále (tj. posun je pevně zvolen). Caesarova šifra spadá do kategorie substitučních šifer (stejný znak je při více vyskytech vždy zašifrován na stejný znak).

Vigenerova šifra Rozšíření Caesarovy šifry, klíč je delší než 1 znak. Klíč je řetězec, který reprezentuje posuny. V případě že vstup je delší než klíč, je klíč perioricky opakován. Vigenerova šifra spadá do kategorie polyalfabetických substitučních šifer (stejný znak může být při více výskytech zašifrován na jiný znak).

Vernamova šifra (*One Time Pad*) Vernamova šifra spadá do kategorie polyalfabetických substitučních šifer a je i dnes nerozluštitelná pokud:

- klíč je delší než vstupní text,
- klíč se nepoužije opakovaně,
- klíč je náhodný.



Obrázek 7.1: Vernamova šifra.

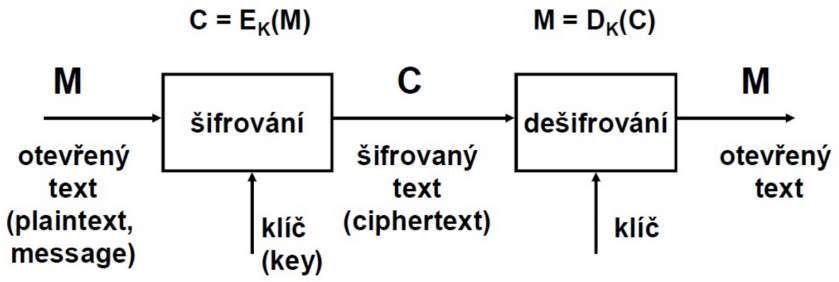
Autoklíč (*autokey*) Šifrování klíčem a když vstupní text je delší než klíč, tak se pokračuje šifrováním otevřeným nebo šifrovaným textem. Lze použít u Vigenerovy nebo Vernamovy šifry.

Symetrická kryptografie Algoritmy používají k šifrování i dešifrování stejný klíč. Výhodou symetrických šifer je jejich nízká výpočetní náročnost. Asymetrické šifry mohou být i stotisíckrát pomalejší. Nevýhodou je nutnost sdílení tajného klíče, takže jedna strana musí klíč vygenerovat a potom ho bezpečným způsobem předat druhé straně.

Typy útoků

- Ciphertext Only Attack (COA) – Útočník zná pouze zašifrovaný text a snaží se zjistit klíč nebo otevřený text. Nejčastěší případ.
- Known Plaintext Attack (KPA) – Útočník zná zašifrovaný text a otevřený text a snaží se zjistit klíč.
- Chosen Plaintext Attack (CPA) – Útočník zná to co v KPA a navíc si text může zvolit.

Útok silou Pří útoku silou (*brute force*) zkouší útočník všechny teoreticky možné klíče, dokud nenajde ten správný.



Obrázek 7.2: Princip kryptografie, podle typu klíčů dělíme na symetrickou (tajný klíč) a asymetrickou (veřejný klíč, soukromý klíč).

Bezpečný algoritmus V moderní kryptografii je nepřijatelné utajování algoritmů (*security by obscurity*) – předpokládáme, že útočník zná šifrovací algoritmus. Bezpečnost musí záviset pouze na utajení klíče (Kerckhoffuv princip, *security by design*). Symetrický algoritmus je považován za bezpečný, pokud neexistuje rychlejší útok než útok silou.

Délka klíče Dnes je považováno 80 bitů a více za dostatečné. Typicky se délka zaokrouhluje na mocninu 2 (typicky 128b). Klíče symetrických algoritmů jsou kratší než asymetrických. Konkrétně: DES – 56b, 3DES – 112, AES – variabilní.

Využití Symetrická kryptografie je vhodná pro šifrování většího objemu dat. Narozdíl od asymetrické, která je pro tento účel příliš pomalá. Proto např. HTTPS využívá asymetrickou kryptografii pro výměnu symetrických klíčů a poté symetrickou kryptografii pro šifrování provozu.

Vlastnosti moderní kryptografie Symetrická kryptografie zaručuje všechny následující, kromě nepopiratelnosti – více entit má k dispozici klíč.

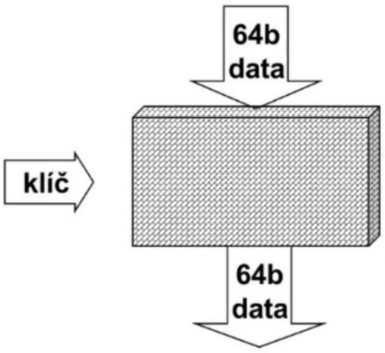
- Důvernost – Utajení informace. Bez znalosti klíče, není možné data číst.
- Autentizace – Prokázání, že zprávu skutečně poslal odesílatel a nikoliv útočník, který se za odesílatele vydává.
- Integrita – Prokázání, že nikdo nemohl data po cestě od odesílatele k příjemci změnit. Ochrana proti neoprávněné, neodhalené modifikaci zprávy.
- Nepopiratelnost – Pokud odesílatel data poslal, nemůže tuto skutečnost popřít.

7.3 Blokové šifry

Blokové šifry šifrují data po blocích pevně stanovené délky (64b, 128b, 256b, ...). Pokud je dat více, rozdělí se na více bloků, přičemž do zbylého místa v posledním je umístěno zarovnání *padding* (informace o délce zarovnání může být obsažena v posledním bytu). Příklady blokových šifer:

- Feistelova šifra (spíše princip)
- Data Encryption Standard (DES)
- Triple Data Encryption Algorithm (3DES)
- International Data Encryption Algorithm (IDEA)

- Blowfish
- Tiny Encryption Algorithm (TEA)
- Advanced Encryption Standard (AES)



Obrázek 7.3: Princip blokových šifer.

7.3.1 Feistelova šifra

Feistelova šifra (Feistelův princip) je koncept šifrování, který konkrétní algoritmy využívají. Jedná se o substituční-permutační síť. Vstupní blok je rozdělen na dvě poloviny L a R, výpočet výstupu pak vypadá následovně.

$$L_i = R_{i-1} \tag{7.1}$$

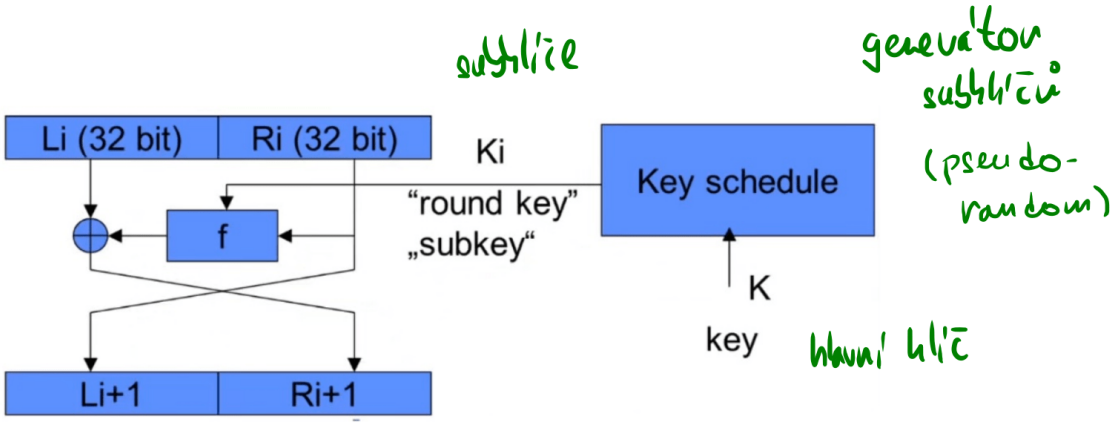
$$R_i = L_{i-1} \oplus F(R_{i-1}, K_i) \tag{7.2}$$

Funkce F F je funkce, na kterou Feistelova šifra neklade žádné požadavky. Jednotlivé algoritmy, využívající Festelovu šifru, funkci samy definují. Požadavky na funkci F, aby algoritmus byl bezpečný:

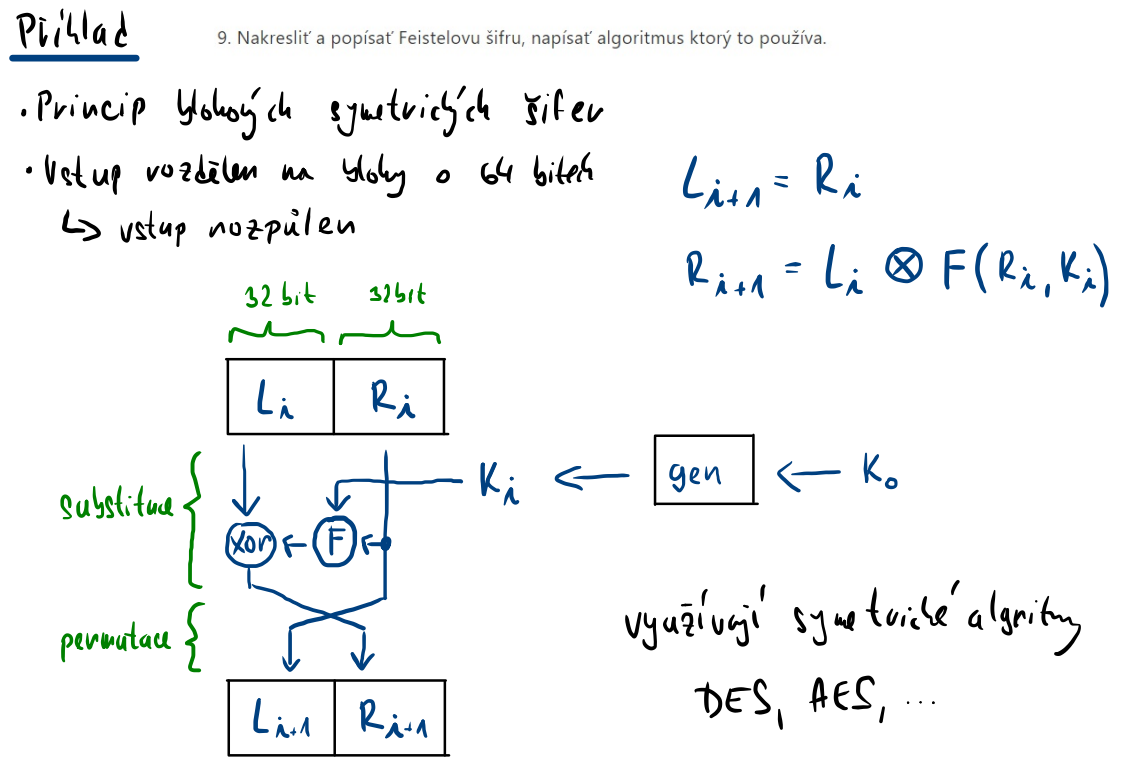
- skrytí vlastností zprávy;
- skrytí vlastností zprávy.

Subklíč K je tzv. subklíč, který je generován typicky nějakým pseudonáhodným generátorem na základě inicializačního klíče (hlavní).

Dešifrování Dešifrování se provádí stejným způsobem, pouze pořadí subklíčů je opačné.



Obrázek 7.4: Jeden krok opakování (Feistelův krok) vizuálně.



Obrázek 7.5: Feistelova šifra – příklad a rekapitulace.

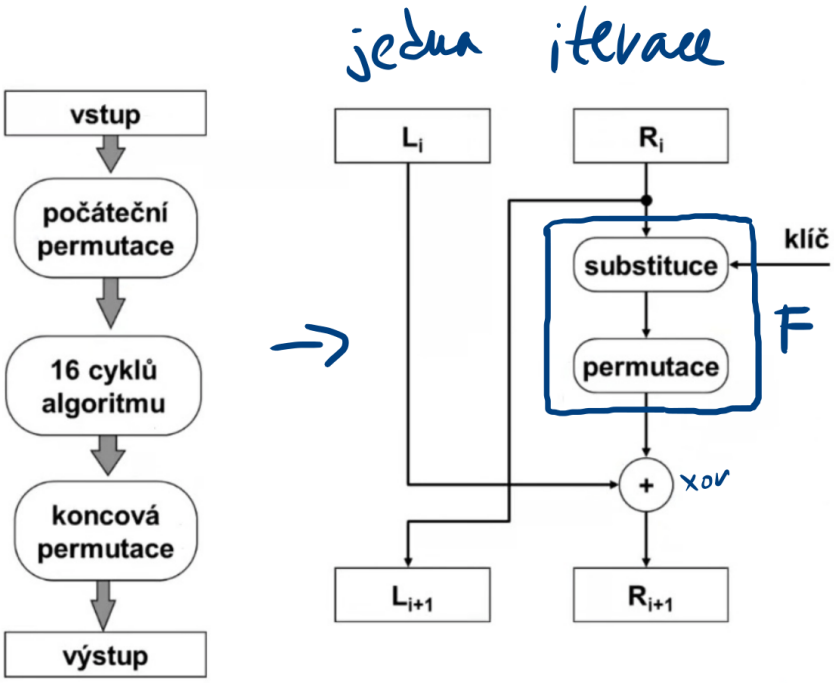
7.3.2 Data Encryption Standard (DES)

DES byl první algoritmus s veřejnou specifikací (*security by design*). Využívá princip Feistelovy šifry – 16 kol. Dodatečně přidává na začátek a konec permutaci navíc. Klíč je dlouhý 64b (resp. 56 významových bitů a 8 paritních).

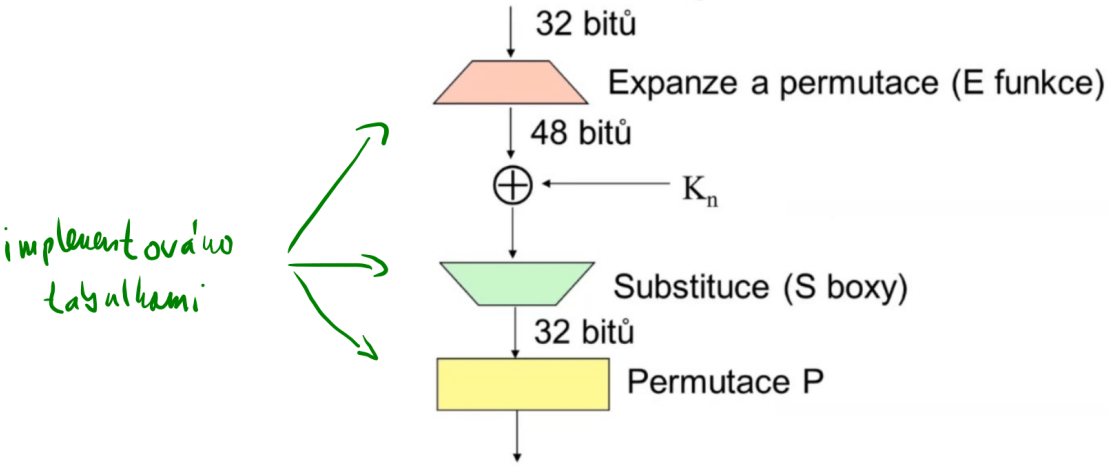
Slabiny

- 56 bitový klíč je příliš krátky a je možný útok silou.

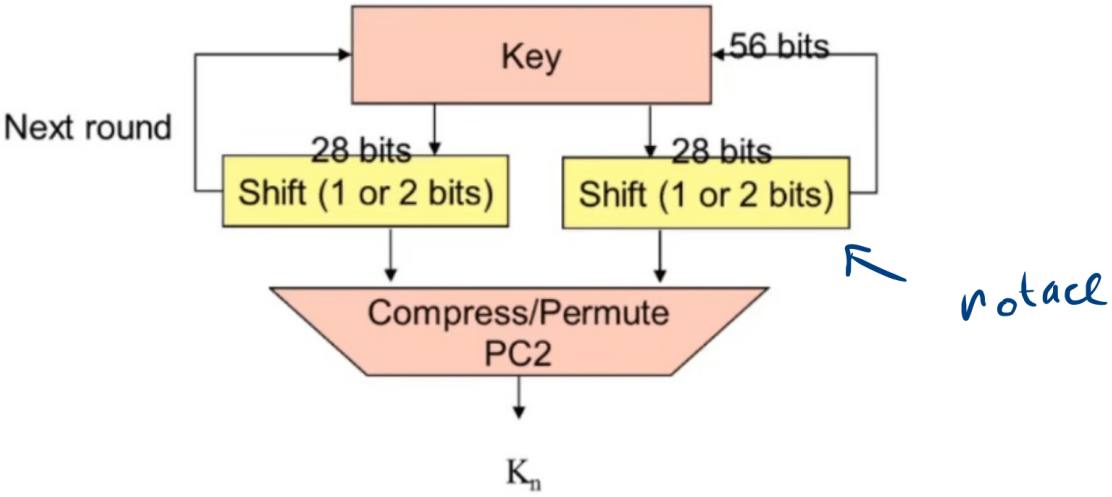
- Rozdílná velikost bloku a klíče (zvláštnost).
- Existence slabých a poloslabých klíčů.
- Není jasné proč zrovna 16 iterací a zda je to dostatečné.



Obrázek 7.6: DES – Schéma fungování algoritmu.



Obrázek 7.7: DES – funkce F.



Obrázek 7.8: DES – generování subklíčů.

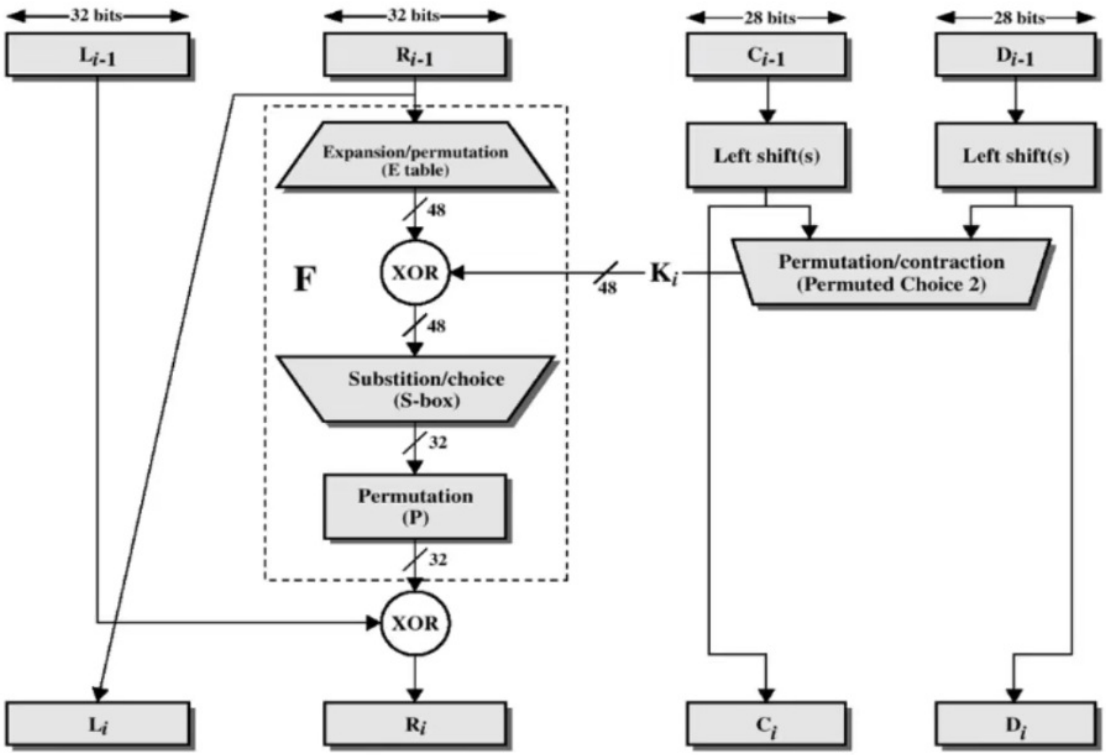
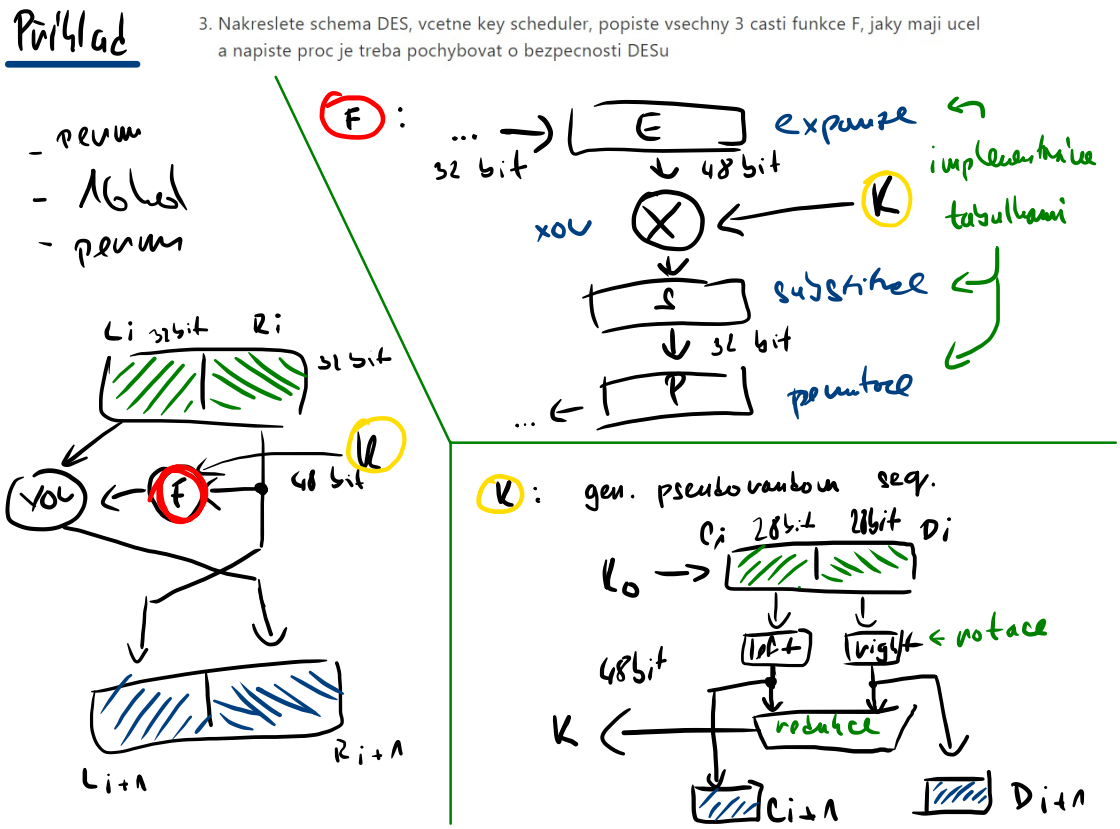


Figure 2.4 Single Round of DES Algorithm

Obrázek 7.9: DES – jedno kolo algoritmu. *Left shift* je ve skutečnosti bitová rotace.



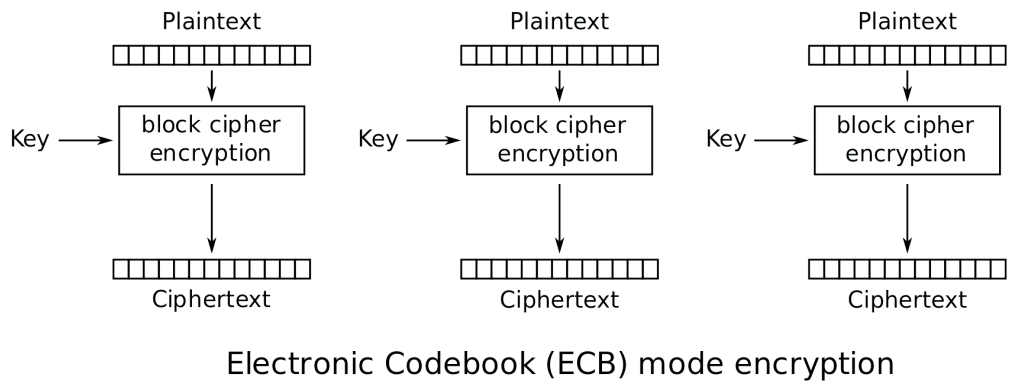
Obrázek 7.10: DES – příklad a rekapitulace.

7.4 Provozní režimy činnosti blokových šifer

Jak použít blokové šifry abychom byli schopni šifrovat data delší než jeden blok?

7.4.1 Electronic Code Book (ECB)

ECB („kódová kniha“) je výchozí *naivní* režim. Bloková šifra se při něm přímo aplikuje nezávisle na jednotlivé bloky, tedy při daném klíči odpovídá stejnému bloku otevřeného textu stejný blok šifrového textu. To má nežádoucí důsledky z hlediska bezpečnosti, v datech zůstane původní struktura, např. šifrovaný obrázek je rozpoznatelný.

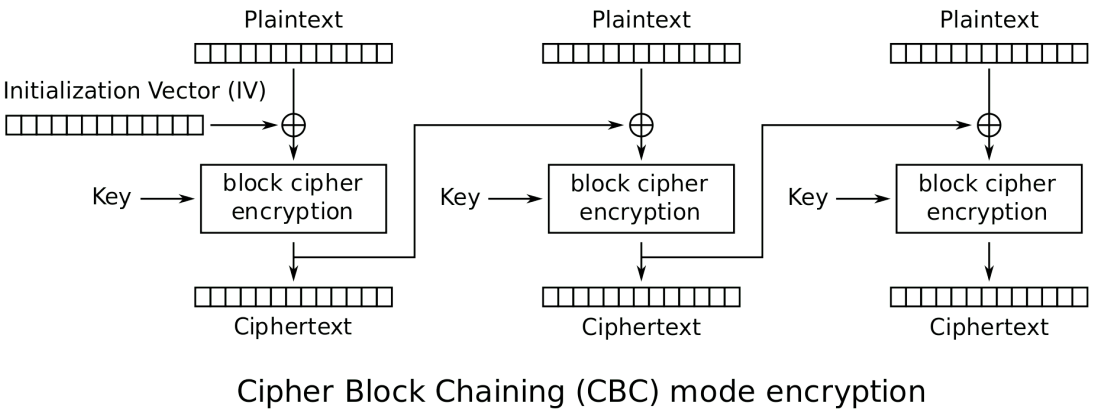


Obrázek 7.11: Ukázka režimu ECB.

7.4.2 Cipher Block Chaining (CBC)

V režimu CBC („řetězení šifrových bloků“) je každý blok před šifrováním xorován zašifrovaným předchozím blokem a první blok je xorován inicializačním vektorem. Tento režim je široce používán. Nevýhody plynou ze zřetězené závislosti (šifrovaný blok závisí na všech předcházejících): Šifrování nelze paralelizovat a při poškození šifrového bloku nelze dešifrovat ani blok přímo následující. Dešifrování paralelizovat lze.

$$\begin{aligned} C_i &= E_K(P_i \oplus C_{i-1}) \\ P_i &= D_K(C_i) \oplus C_{i-1} \\ C_0 &= IV \end{aligned} \tag{7.3}$$

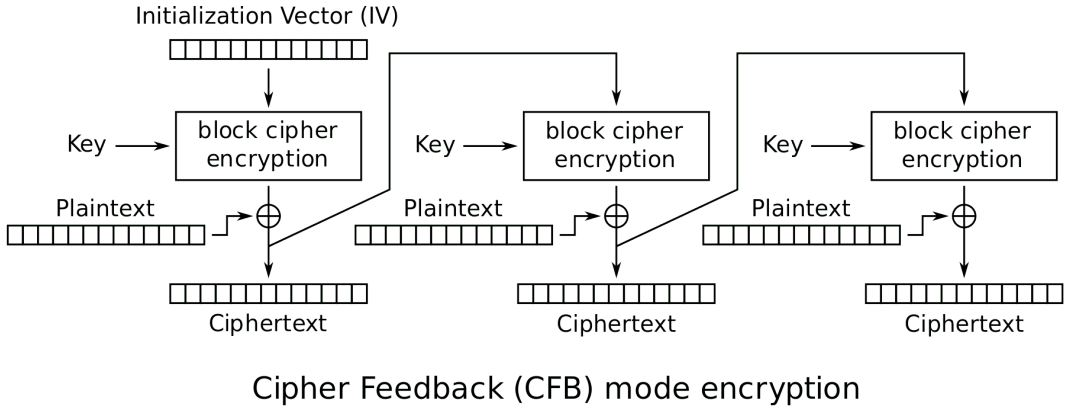


Obrázek 7.12: Ukázka režimu CBC.

7.4.3 Cipher Feedback (CFB)

Režim CFB (*šifrová zpětná vazba*) se liší oproti CBC v prohození pořadí operací xor a šifrování – nejprve se zašifruje předchozí šifrovaný blok (resp. inicializační vektor) a výsledek se xoruje s otevřeným blokem. Toto prohození má významné implementační dopady: díky symetrii operace XOR vypadá dešifrovací funkce obdobně jako šifrovací. Šifruje pomaleji než CBC. Vstup není nutné zarovnávat. Plynou stejné nevýhody jako pro CBC.

$$
\begin{aligned}
C_i &= E_K(C_{i-1}) \oplus P_i \\
P_i &= E_K(C_{i-1}) \oplus C_i \\
C_0 &= IV
\end{aligned}
\tag{7.4}
$$



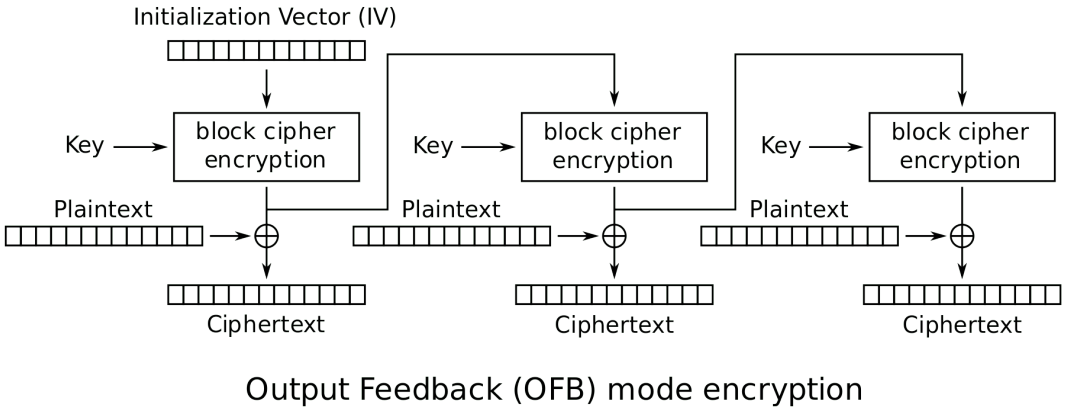
Obrázek 7.13: Ukázka režimu CFB.

7.4.4 Output Feedback (OFB)

Režim OFB (*výstupní zpětná vazba*) se liší od CFB pouze v tom, kde bere zpětnou vazbu. Šifrování probíhá pouhým xorováním otevřeného bloku s heslem, které je v každém kroku zašifrováno použitou blokovou šifrou. První blok hesla je získán zašifrováním inicializačního vektoru. Režim převádí blokovou šifru na synchronní proudovou šifru.

Slabina Celý blok encryption je pouze generátor pseudonáhdon posloupnosti (je nezávislá na otevřeném nebo šifrovaném textu). To umožňuje Known Plaintext Attack. Z toho plyne, že jedním klíčem není bezpečné šifrovat více než jednu zprávu.

$$
\begin{aligned}
C_i &= E_K(C_{i-1}) \\
P_i &= P_i \oplus C_i \\
C_0 &= IV
\end{aligned}
\tag{7.5}
$$

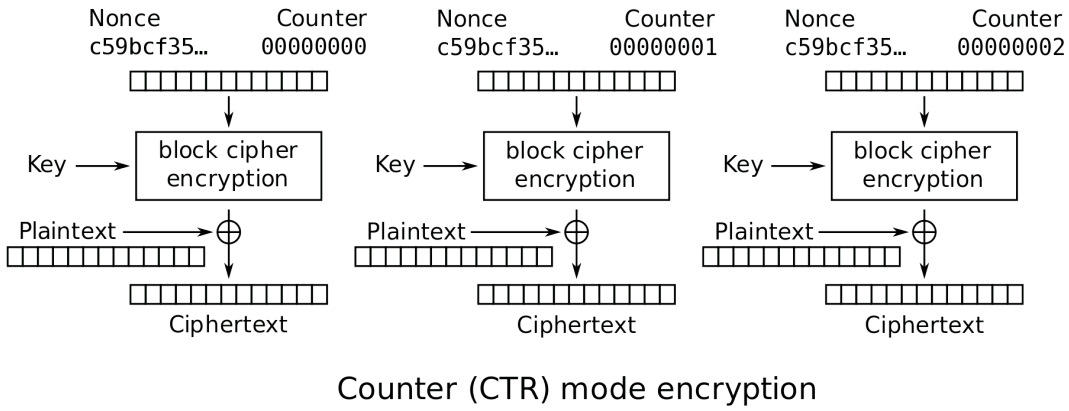


Obrázek 7.14: Ukázka režimu OFB.

7.4.5 Counter (CTR)

Režim CTR („čítačový režim“) převádí stejně jako OFB blokovou šifru na synchronní proudovou. Heslo, se kterým se blok otevřeného textu xoruje, je však získáno zašifrováním čítače, který se každou iteraci zvětšuje o pevně danou hodnotu, zpravidla o 1. Obsah čítače je opět před šifrováním nastaven inicializačním vektorem. Každý blok je šifrován nezávisle na ostatních, díky tomu je možné paralelizovat.

$$\begin{aligned} CTR_i &= CTR_{i-1} + 1 \\ P_i &= P_i \oplus E_k(CTR_i) \\ CTR_0 &= IV \end{aligned} \tag{7.6}$$



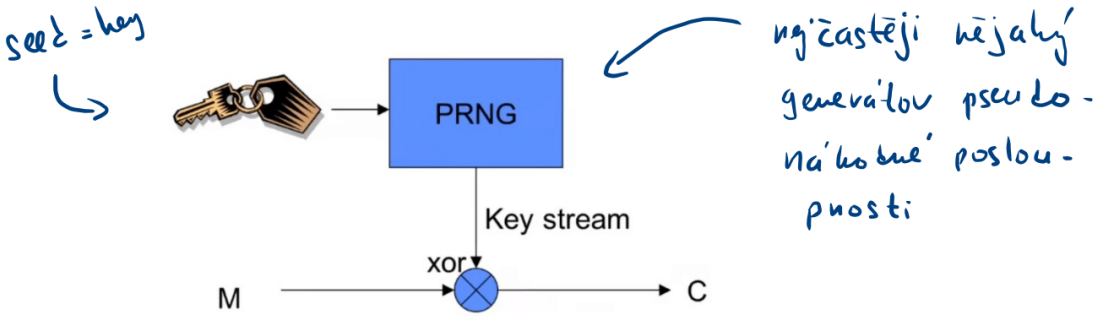
Obrázek 7.15: Ukázka režimu CTR.

7.5 Proudové šifry

Proudové šifry šifrují data jako *proud* (stream), nejčastěji po jednotlivých bytech. Dešifrování vždy probíhá stejným způsobem. Proudové šifry jsou rychlejší než blokové šifry a pro implementaci potřebují jednodušší hardware.

Problémy

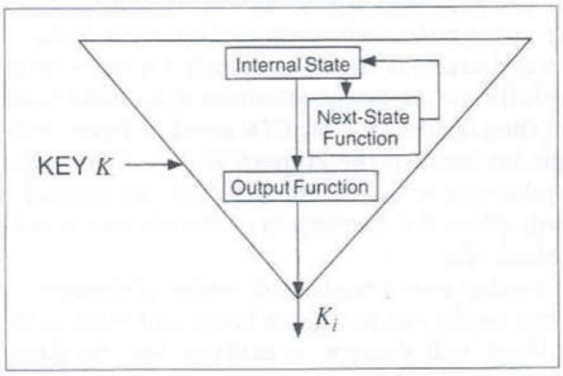
- Nezajišťují samy o sobě integritu.
- Na rozdíl od blokových šifer jsou náchylnější ke kryptoanalytickým útokům, pokud jsou nevhodně implementovány (počáteční stav nesmí být použit opakovaně) – „problém s inicializačním vektorem“.



Obrázek 7.16: Princip proudových šifer.

7.5.1 Rozdělení proudových šifer

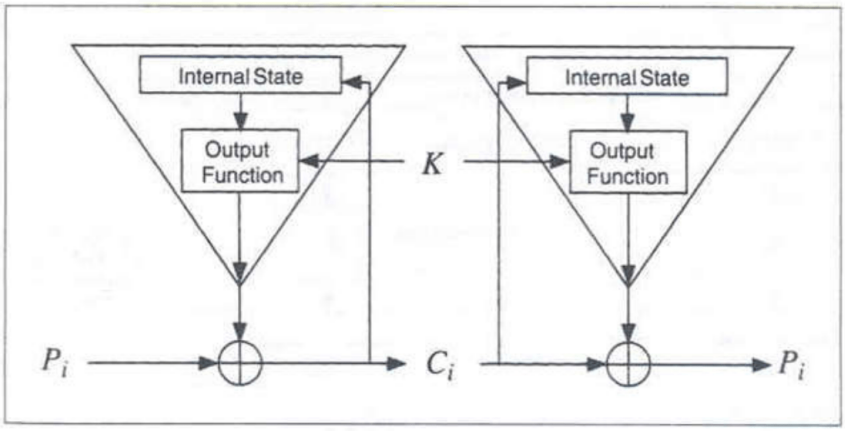
Synchronní proudové šifry Proud pseudonáhodných čísel *key stream* je generován nezávisle na vstupním textu nebo zašifrované zprávě. Poté dochází ke kombinaci vygenerovaných čísel se vstupujícím textem (k zakódování) nebo se šifrovaným textem (k dekódování). Nejběžnější formou kombinace keystreamu a vstupního textu je použití operace XOR. Např.: Vernamova šifra, DES v režimu OFB. Pokud se průběhu dešifrování něco ztratí, je konec.



Obrázek 7.17: Princip synchronní proudové šifry.

Samosynchronizující proudové šifry Proud pseudonáhodných čísel *key stream* závisí na pevném počtu předcházejících bytů šifrovaného (nebo otevřeného) textu. To znamená, že se šifra dokáže po chybě sama *zotavit* (resynchronizovat)[1]. Např.: Vigenere Autokey, DES v řežimu CFB.

[1]Dnes se nesnážíme dešifrovat poškozená data, pokud nastane chyba v přenosu, vyžádáme si data znovu.



Obrázek 7.18: Princip samosynchronizující proudové šifry.

7.5.2 Generátory PRNG

Generátory PRNG (*pseudo-random number generator*) generují pseudo-náhodnou posloupnost (*key stream*) z malého klíče (*seed*).

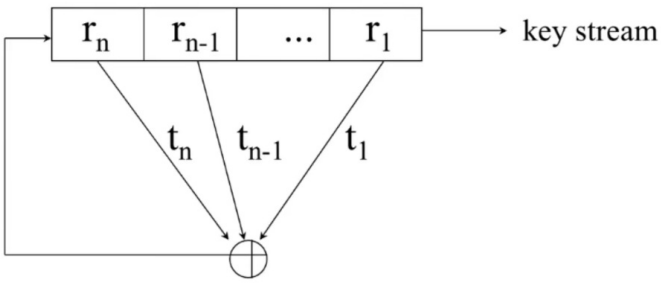
Blokové šifry v režimu OFB Blokové šifry v režimu OFB jsou pomalé.

Linear Feedback Shift Registers (LSFR) LFSR (posuvný registr s lineární zpětnou vazbou) je posuvný registr, jehož výstup je lineárně závislý na jeho předchozích výstupech a stavu. Mějme

- posuvný registr $R = (r_1, r_2, \ldots, r_n)$,
- sekvenci zpětných vazeb $T = (t_1, t_2, \ldots, t_n)$.

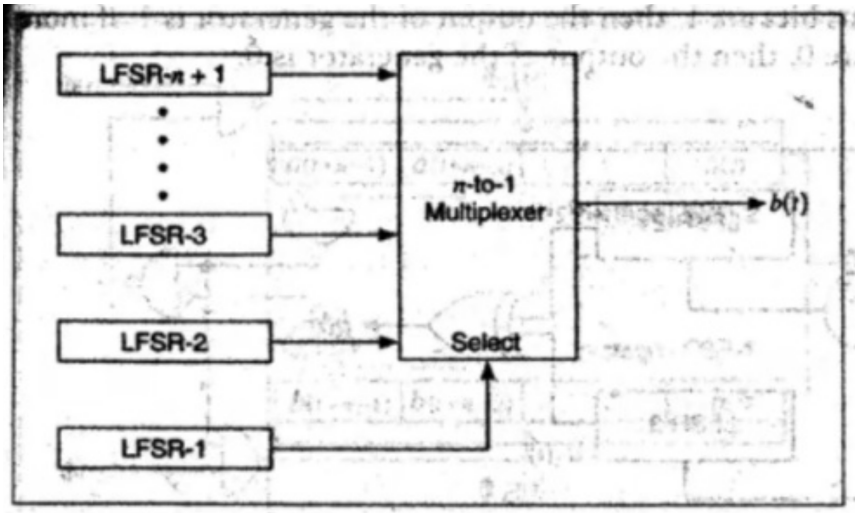
Alternativně lze zapsat polynomem:

$$T(x) = x^n + x^{n-1} + \ldots + x + 1$$



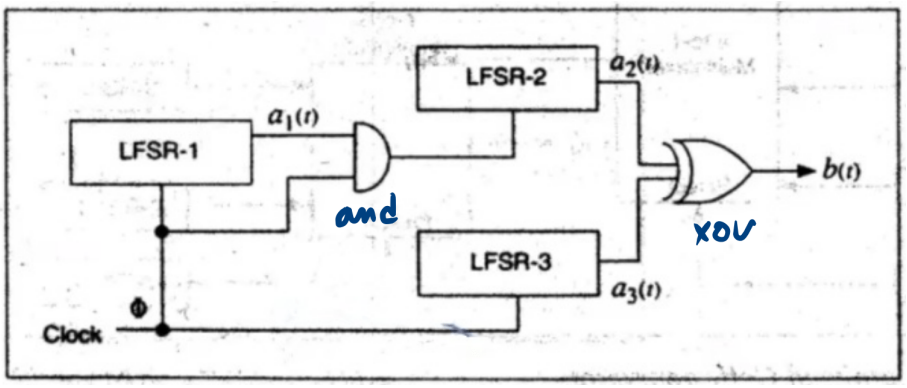
Obrázek 7.19: Příklad LFSR.

Geffe generátor Geffe generátor (kombinovaný generátor) je využití 2 a více LFSR propojených multiplexorem.



Obrázek 7.20: Příklad Geffe generátoru.

Stop and Go generátor Stop and Go generátor je několik LSFR s různým zdrojem hodin.



Obrázek 7.21: Příklad Stop and Go generátoru.

Kapitola 8

KRY – Asymetrická kryptografie, vlastnosti, způsoby použití, poskytované bezpečnostní funkce, elektronický podpis a jeho vlastnosti, hybridní kryptografie, algoritmus RSA, generování klíčů, šifrování, dešifrování.

8.1 Metadata

- Předmět: Kryptografie (KRY)
- Přednáška:
  - 6) Asymetrická kryptografie, vlastnosti, způsoby použití, poskytované bezpečnostní funkce.
  - 7) Elektronický podpis a jeho vlastnosti, hybridní kryptografie.
  - 8) Příklady asymetrických algoritmů, RSA.
- Záznam:
  - 2021-03-08
  - 2021-03-22

8.2 Úvod a kontext

Viz. „Úvod a kontext“ v předchozích otázkách z tohoto předmětu.

Asymetrická kryptografie

- V asymetrické kryptografii se používají páry klíčů (soukromý a veřejný). Soukromý je používán k dešifrování, resp. vytvoření digitálního podpisu. Veřejný je používán k šifrování, resp. ověření digitálního podpisu.

- Každý uživatel generuje svůj pár klíčů. Veřejný klíč je zveřejněn (znají ho všichni), sourkomý je držen v tajnosti (zná ho pouze vlastník).
- Všechny asymetrické algoritmy jsou blokové.
- Asymetrické algoritmy jsou pomalejší než symetrické.

Způsoby použití Asymetrická kryptografie lze využít k:

- šifrování,
- digitálnímu podepisování,
- pro výměnu symetrického klíče (*key exchange*).

Vlastnosti Vlastnosti symetrické a asymetrické kryptografie[1].

	Důvěrnost	Autentizace	Integrita	Nepopiratelnost
Symetrická	ano	?	?	ne
Asymetrická - šifrování	ano	?	?	ne
Asymetrická - podepisování	ne	ano	ano	ano
Asymetrická - kombinace	ano	ano	ano	ano

```
# Odesilatel (A):
msg = encrypt(msg, SK_A) # necht msg je zprava k~odeslani
msg = encrypt(msg, PK_B)
send(msg_2)

# Prijemce (B):
msg = receive()
msg = decrypt(msg, SK_B)
msg = decrypt(msg, PK_A)
```

Výpis 8.1: Kombinace klíčů obou stran u asymetrický kryptografie. Pořadí operací může být i opačné.

Digitální podpis Vytvoření digitálního podpisu konkrétních dat pomocí soukromého klíče podepisatele. Každý kdo zná veřejný klíč podepisatele, může pravost podpisu ověřit. Digitální podpis zajišťuje autentizaci, itegritu a nepopiratelnost.

Algoritmy Algoritmy asymetrické kryptografie se nedají *vymyslet*, musí se objevit. Jsou založeny na těžkých matematických problémech.

- Problém batohu (*knapsack problem*) – MH (Merkle-Hellman)
- Faktorizace čísel – RSA (Rivest-Shamir-Adleman)
- Diskrétní logaritmus – DSA (Digital Signature Algorithm), DH (Diffie-Hellman)
- Eliptické křivky – ECDSA, ECDH

[1] Otazníky – částečně, za předpokladů, ...

Problém batohu Problém batohu je NP-úplný problém kombinatorické optimalizace. Nechť $x_1, x_2, \ldots, x_n$ je množina objektů, každý objekt má svoji cenu v_i a svoji hmotnost w_i, dále mějme batoh, který má kapacitu W. Cílem je vybrat takovou množinu objektů, jejichž hmotnost je menší nebo rovna W a má nejvyšší možnou cenu[2]. Formálně chceme maximalizovat sumu

$$\sum_{i=1}^{n} v_i \cdot x_i$$

, při splnění

$$\sum_{i=1}^{n} w_i \cdot x_i \leq W$$

, kde $x_i \in x_1, x_2, \ldots, x_n$.

Faktorizace čísel Faktorizace čísel označuje problém rozložení čísla na součin menších čísel, v nejčastější podobě pak rozklad celého čísla na součin prvočísel.

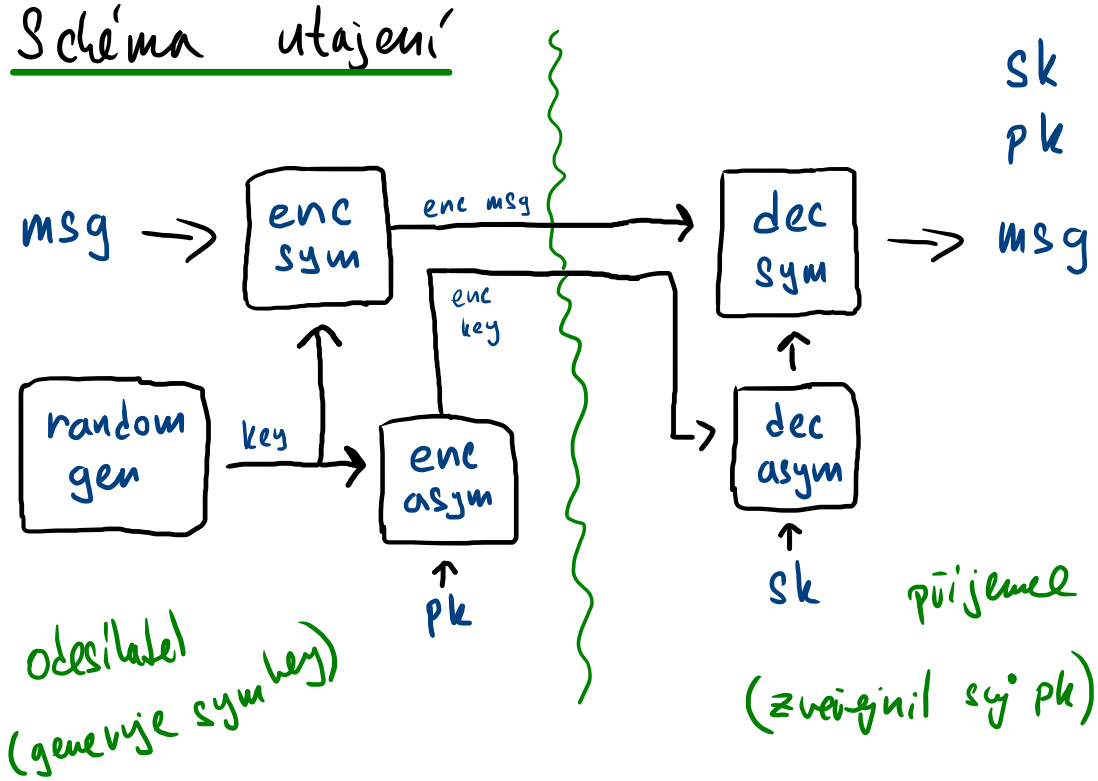
Diskrétní logaritmus Nechť p, g, k, Y jsou přirozená čísla, pro něž platí $Y \equiv g^k \bmod p$. Potom každé číslo k odpovídající uvedené rovnici nazveme diskrétní logaritmus o základu g z Y vzhledem k modulu p. Tato definice nedefinuje číslo k jednoznačně, proto se někdy upravuje tak, že ze všech možných diskrétních logaritmů ve smyslu předchozí definice se vybere ten nejmenší.

Eliptické křivky Jedná se o matematický aparát, na kterém aplikujeme různé algoritmy (DSA, DH).

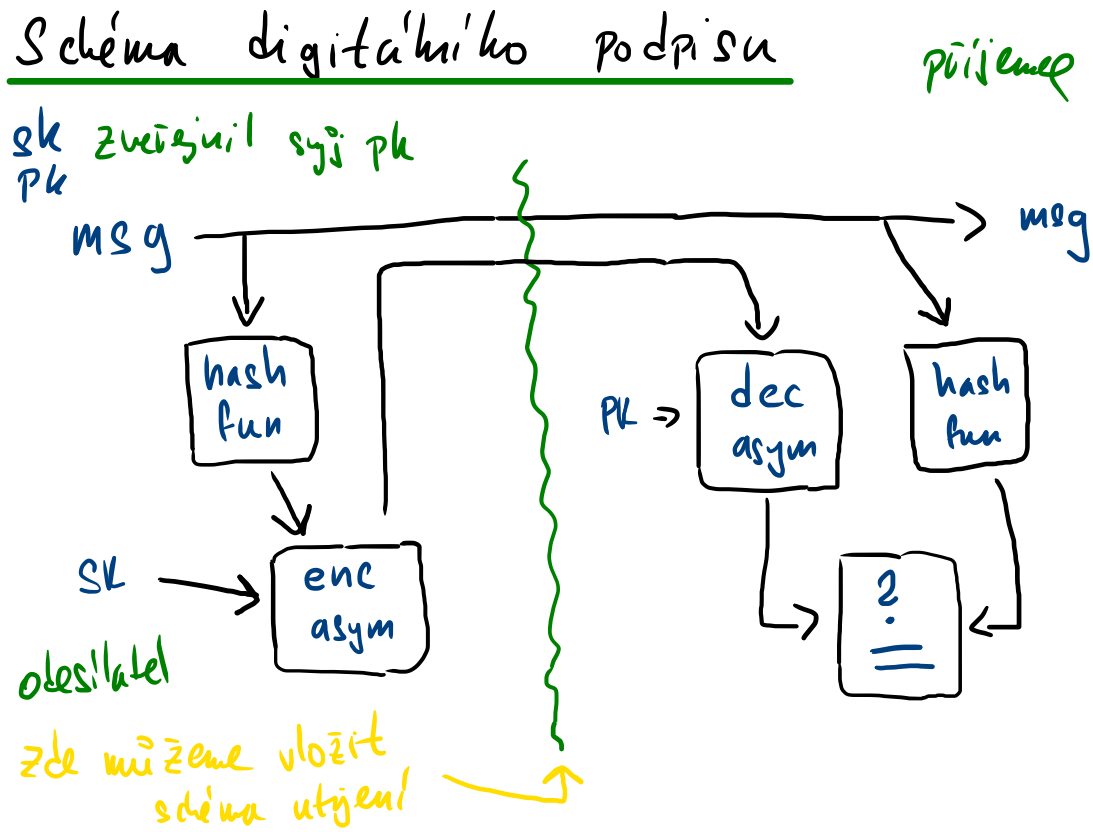
8.3 Hybridní kryptografie

Hybridní kryptografie je kombinace symetrické a asymetrické kryptografie, ve které jsou využity přednosti obou (symetrická – rychlá, ale potřeba stejný klíč; asymetrická – pomalá, ale dva klíče). Asymetrická je využita pro bezpečné zaslání symetrického klíče.

[2] Problém má více obdobných variant.



Obrázek 8.1: Schéma utajení hybridní kryptografie.



Obrázek 8.2: Schéma digitálního podpisu hybridní kryptografie.

8.4 RSA

Algoritmus RSA (Rivest-Shamir-Adleman) lze použít jak pro šifrování dat pro digitální podepisování. Je založen na problému faktorizace velkých čísel.

Klíče Klíče se skládají z:

- p, q – dvě náhodná soukromá prvočísla,
- n – veřejný modul ($n = p \cdot q$),
- e – veřejný exponent ($e < \Phi(n) \ \wedge \ GCD(\Phi(n), e) = 1$), typicky 3 nebo $2^{16} + 1$[3],
- d – soukromý exponent,
- musí platit vztah: $e \cdot d \bmod \Phi(n) = 1$.

Veřejný klíč $PK = (n, e)$, soukromý klíč $SK = (n, d)$.

Postup generování Postup generování klíčů:

1. vygenerovat prvočísla p a q,
2. spočítat modul $n = p \cdot q$,
3. spočítat $\Phi(n) = (p - 1) \cdot (q - 1)$,

[3] GCD – největší společný dělitel

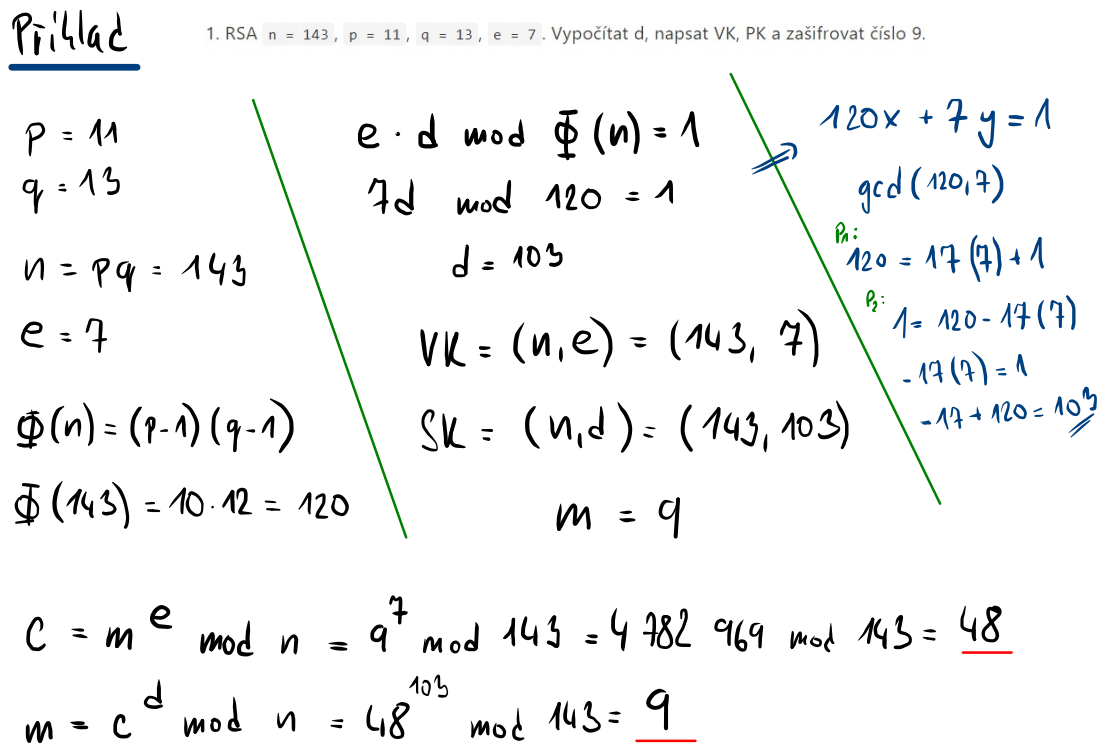
4. zvolit veřejný exponent $e < \Phi(n) \ \wedge \ GCD(\Phi(n), e) = 1$,
5. spočítat soukromý exponent d tak, že platí $e \cdot d \bmod \Phi(n) = 1$.

Šifrování a dešifrování Mějme zprávu m reprezentovanou jako celé číslo a zašifrovanou zprávu c reprezentovanou také jako celé číslo. Digitální podpis se vytváří stejným způsobem, pouze se prohodí exponenty.

$$c = m^e \bmod n \tag{8.1}$$

$$m = c^d \bmod n \tag{8.2}$$

Útoky a slabiny Pokud útočník rozloží číslo n na činitele p a q, tak může dopočítat soukromý klíč. Pokud útočník uhádně hodnotu $(p-1)\cdot(q-1)$, tak může dopočítat soukromý klíč. Šifrování malých čísel je zranitelné, proto se používá „předzpracování“ – zarovnání na X bitů (2048).



Obrázek 8.3: Příklad RSA.

Kapitola 9

KRY – Hašovací funkce, klíčovaný haš a MAC a jejich použití a vlastnosti.

9.1 Metadata

- Předmět: Kryptografie (KRY)
- Přednáška:
  - `KRY04_Asym_MNG.pdf`
- Záznam:
  - 2021-03-22

9.2 Úvod a kontext

Viz. „Úvod a kontext“ v předchozích otázkách z tohoto předmětu.

Hashovací funkce Hashovací funkce je funkce (resp. algoritmus) pro převod vstupních dat do (relativně) malého čísla. Výstup hashovací funkce se označuje otisk, *fingerprint*, *digest* či *hash*. Jsou jednosměrné a odolné proti kolizím (viz vlastnosti).

Obecné vlastnosti Hashovací funkce by měla:

- Být aplikovatelná na argument o libovolné velikosti.
- Mít výstyp konstantní délky.
- Dokázat spočítat výstup rychle.

Neklíčované hashovací funkce Hashovací funkce má pouze jeden argument – data. Např. MD2, MD4, MD5, SHS, SHA1, SHA2, SHA3.

$$f(data) \rightarrow hash$$

Klíčované hashovací funkce Hashovací funkce má dva argumenty – data a klíč. Také se jim někdy říká MAC (*message authentication code*).

$$f(data, key) \rightarrow hash$$

9.3 Kryptografická odolnost hashovacích funkcí

Vlastnosti z hlediska odolnosti Hashovací funkce by z hlediska kryptografické odolnosti měly splňovat:

- *First preimage resistance* – Pro konkrétní y je výpočetně nezvládnutelné najít takové x, aby platilo $h(x) = y$. Útočník má k dispozici konkrétní hash, a snaží se pro něho nalézt zprávu.
- *Second preimage resistance* – Pro konkrétní x je výpočetně nezvládnutelné najít takové x', aby platilo $h(x) = h(x')$. Útočník má k dispozici konkrétní zprávu (nemůže si ji zvolit), ke které se snaží nalézt jinou zprávu, která bude mít stejný hash.
- *Collision resistance* – Je výpočetně nezvládnutelné najít libovolnou dvojici x, x' takovou, aby platilo $x \neq x'$ a $h(x) = h(x')$. Útočník si může zvolit libovolnou zprávu, ke které se snaží nalézt jinou zprávu, která bude mít stejný hash. Pokud platí *collision resistance*, tak platí i *second preimage resistance*.

Narozeninový problém V teorii pravděpodobnosti je narozeninový problém úloha vypočítat minimální početnost skupiny lidí, ve které je alespoň 50% pravděpodobnost nalezení dvojice se stejným datem narození. Narozeninovým paradoxem je pak označována skutečnost, že tento počet (23) je mnohem menší než intuitivní odhad. Výsledek je intuitivnější, když uvážíme, že porovnání narozenin bude provedeno mezi všemi možnými dvojicemi jedinců. Při počtu 23 jedinců je třeba uvažovat $(23 \cdot 22)/2 = 253$ dvojic, což je více než polovina počtu dnů v roce $(182,5)$.

Jednodušší je nejprve spočítat jev opačný $\bar{p}(n)$, tedy pravděpodobnost, že všech n narozenin je rozdílných. Pro $n > 365$ je 1, jinak:

$$\begin{aligned}\bar{p}(n) &= 1 \cdot \left(1 - \frac{1}{365}\right) \cdot \left(1 - \frac{2}{365}\right) \cdots \left(1 - \frac{n-1}{365}\right) = \\ &= \frac{365 \cdot 364 \cdots (365 - n + 1)}{365^n} = \\ &= \frac{365!}{365^n (365 - n)!}\end{aligned} \tag{9.1}$$

$$p(n) = 1 - \bar{p}(n) \tag{9.2}$$

Narozeninový útok Mějme hashovací funkci, která má n bitový výstup (celkový počet možných hashů je 2^n). Útočník vytvoří dokument „přátelská dohoda“ a přibližně $2^{n/2}$ sémanticky ekvivalentních verzí (úprava bílých znaků, úprava pořadí celků, jiné formulace, ...). Podobně vytvoří dokument „nepřátelská dohoda“ a přibližně $2^{n/2}$ sémanticky ekvivalentních verzí. S pravděpodobností $0,5$ existuje verze „přátelské dohody“ a „nepřátelské dohody“, které mají stejný hash. Pokud takové verze existují, útočník dá oběti podepsat „přátelskou dohodu“ $\Rightarrow$ existuje validní podpis „nepřátelské dohody“.

Bezpečnostní cíle OWHF Bezpečnostní cíle OWHF (*one way hash function*). Některé protokoly nevyžadují bezkoliznost, proto má smysl řešit i tento případ.

- Vyžadované vlastnosti: *first preimage resistance* a *second preimage resistance*
- Cíl útočníka: vytvořit *first preimage* nebo *second preimage* (oba úkoly jsou stejně těžké)

- Složitost: $O(2^n)$ (n je počet bitů hashe)
- Požadovaná délka: $n \geq 80$

Bezpečnostní cíle CRHF Bezpečnostní cíle CRHF (*collision resistance hash function*).

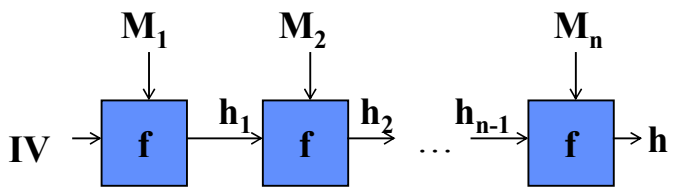
- Vyžadované vlastnosti: *collision resistance*
- Cíl útočníka: vytvořit kolizi
- Složitost: $O(2^{n/2})$ (n je počet bitů hashe) (kvůli narozeninovému útoku)
- Požadovaná délka: $n \geq 160$

Bezpečnostní cíle MAC Bezpečnostní cíle MAC (*message authentication code*).

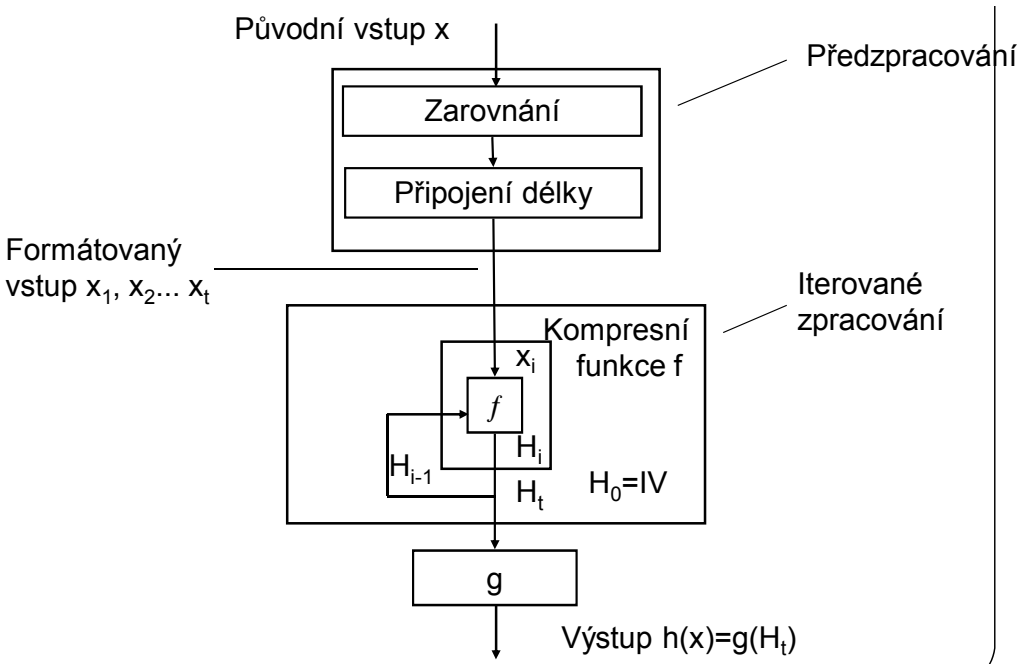
- Vyžadované vlastnosti: *computation resistance*, *key non-recovery*
- Cíl útočníka (útočník si může vybrat):
  - Vytvořit nový hash, který bude odpovídat nové zprávě
  - Nalézt klíč
- Složitost (n je počet bitů hashe, t je počet bitů klíče):
  - Vytvořit nový hash: $O(max(2^{-n}, 2^{-t}))$
  - Nalézt klíč: $O(2^n)$
- Požadovaná délka: $n \geq 64 \ \wedge \ t \geq 64$

9.4 Hashovací funkce neklíčované

Nejčastější způsoby sestrojení hashovací funkce neklíčované jsou založené na principu iterace.



Obrázek 9.1: Schéma iterativní neklíčované hashovací funkce. Zpráva je rozdělena na n částí. f je tzv. kompresní funkce. IV je inicializační vektor, resp. konstanta. h_1 až h_{n-1} jsou mezivýsledky („mezihashe“) a h je výsledný hash.



Obrázek 9.2: Podrobnější schéma iterativní neklíčované hashovací funkce.

Jednotlivé kroky hashovací funkce:

- Předzpracování – Vstupní data jsou rozdělena na bloky o stejné délce. Je provedeno zarovnání posledního bloku. Je připojena informace o délce zprávy.
- Iterativní zpracování – V iteracích se postupně „přihashovávají“ vstupní bloky. Zpětná vazba pomocí stavové proměnné. Uvnitř kompresní funkce, která z delšího vstupu uděla kratší výstup.
- Postzpracování – Volitelný krok.

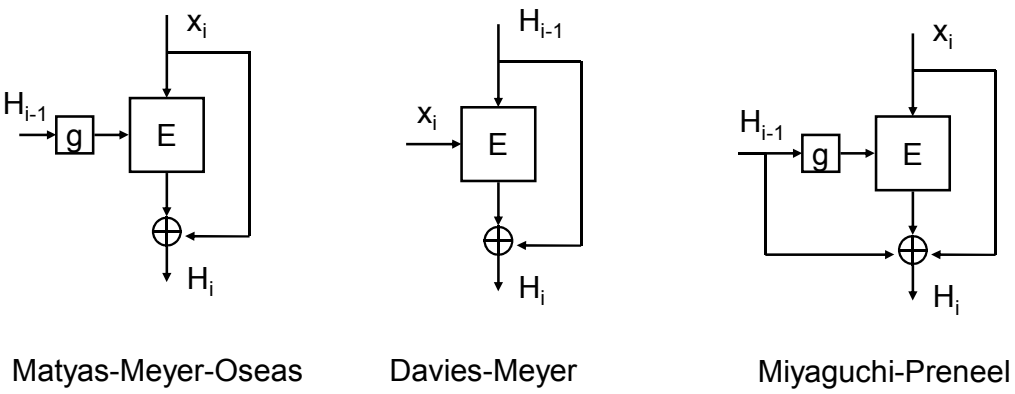
Merkelova meta-metoda Nechť f je kompresní funkce odolná proti kolizím. Hashovací funkce h na principu iterace využívající kompresní funkci f je rovněž odolná proti kolizím.

Merkel-Damgardovo zesílení Pokud je do vstupu hashovací funkce vložena délka zprávy, tak je zajištěno, že žádná zpráva není prefixem jiné zprávy.

Zarovnání Nejednoznačné zarovnání (*ambiguous padding*) – připoj ke zprávě tolik bitů, aby délka zprávy byla násobkem délky bloku. Jednoznačné zarovnání (*unambiguous padding*) – připoj ke zprávě jeden bit a poté proveď nejednoznačné zarovnání.

9.4.1 Hashovací funkce s využitím blokových šifer

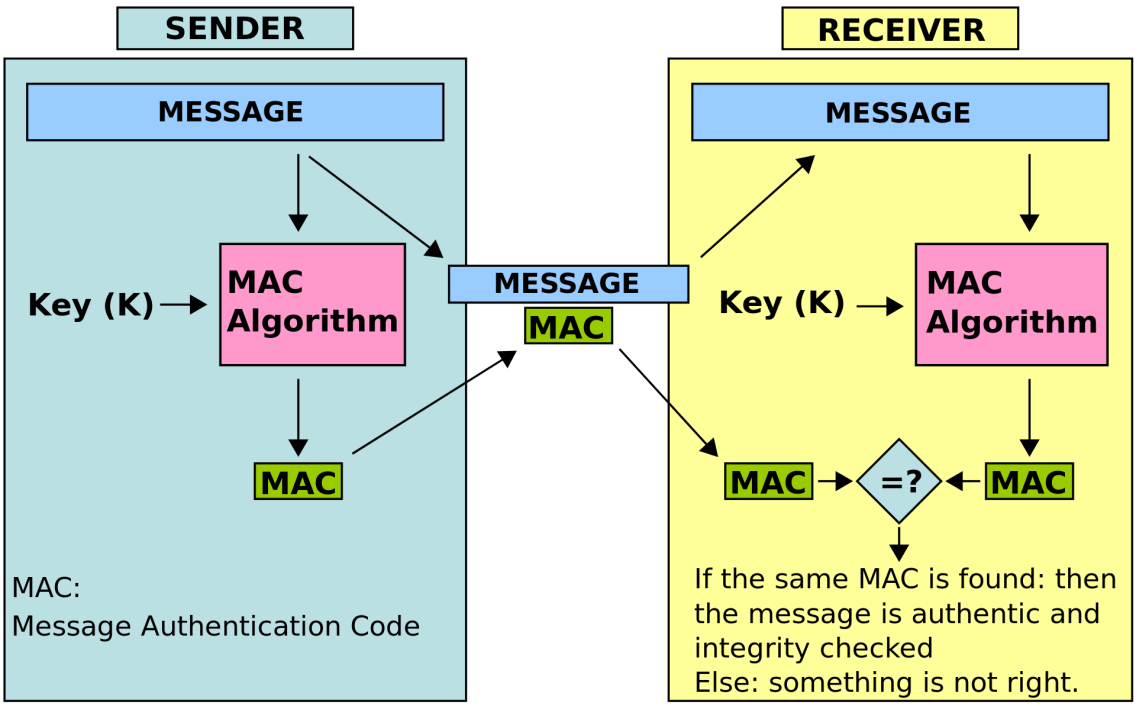
Alternativně lze využít pro konstrukci hashovacích funkcí blokové šifry. Avšak blokové šifry byly navrhovány pro jiný režim činnosti, kterém útočník nezná klíč (a není schopen ho ovlivnit), zná pouze šifrovaný text (ten je schopen ovlivnit). V tomto případě útočník může přimo ovlivňovat hodnoty klíče.



Obrázek 9.3: Ukázka několika možných způsobů využití blokových šifer pro konstrukci kompresní funkce. S využitím iteračního způsobu lze zobecnit pro celou hashovací funkci.

9.5 MAC (*message authentication code*)

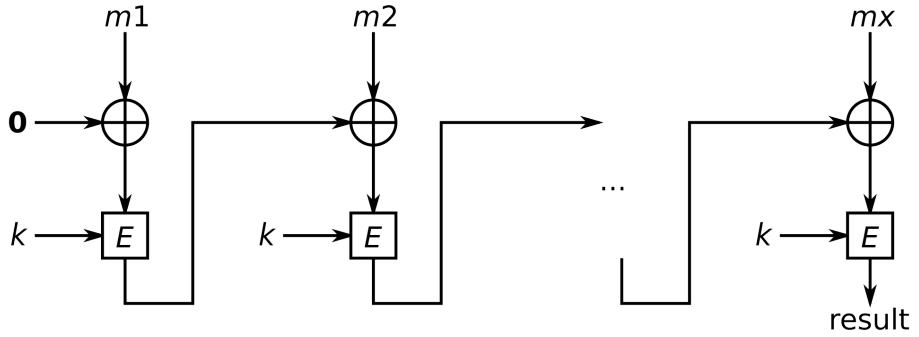
- Rodina hashovacích funkcí h_k, které jsou parametrizovatelné klíčem k.
- Vlasnosti (stejné jako u obecných hashovacích funkcí, pouze rozšířené o klíč):
  - Výstup $h_k(x)$ lze spočítat rychle, pokud je znám klíč k.
  - Jsou výpočetně bezpečné – při znalosti dvojice $(x, h_k(x))$ je výpočetně nemožné spočíst novou dvojici $(x', h_k(x'))$, pro $x \neq x'$, pokud není znám klíč.
- Využití: zajištění autentizace a integrity (nepopiratelnost zajistit nedokáže).



Obrázek 9.4: Schéma použití MAC. Pokud je stejný MAC výpočítán na straně příjemce, tak má jistotu, že zpráva nebyla po cestě změněna a že zprávu poslal skutečně odesílatel.

9.5.1 Sestrojení MAC pomocí blokové šifry v CBC

Pro sestrojení MAC hashovací funkce je využita symetrická bloková šifra v režimu CBC (*cipher block chaining*, šifrová zpětná vazba). Rozdíl oproti CBC šifrování spočívá v tom, že mezivýsledky se zahazují a pracuje se až s posledním blokem. Z něho se vezme určitý počet posledních bitů (podle požadované délky hashe – 32, 48, 64) a ten tvoří výsledný hash (MAC).



Obrázek 9.5: Ukázka sestrojení MAC funkce pomocí symetrické blokové šifry v režimu CBC.

Proč stačí výrazně menší délka? Protože klíč. Útočník sice může vyzkoušet všechny možné hashe, ale bez znalostí klíče, nezjistí, který je ten správný.

9.5.2 Sestrojení MAC pomocí neklíčované hashovací funkce

Pro sestrojení MAC hashovací funkce je využita neklíčovaná hashovací funkce. Klíč je připojen ke zprávě a je použita standardní hashovací funkce.

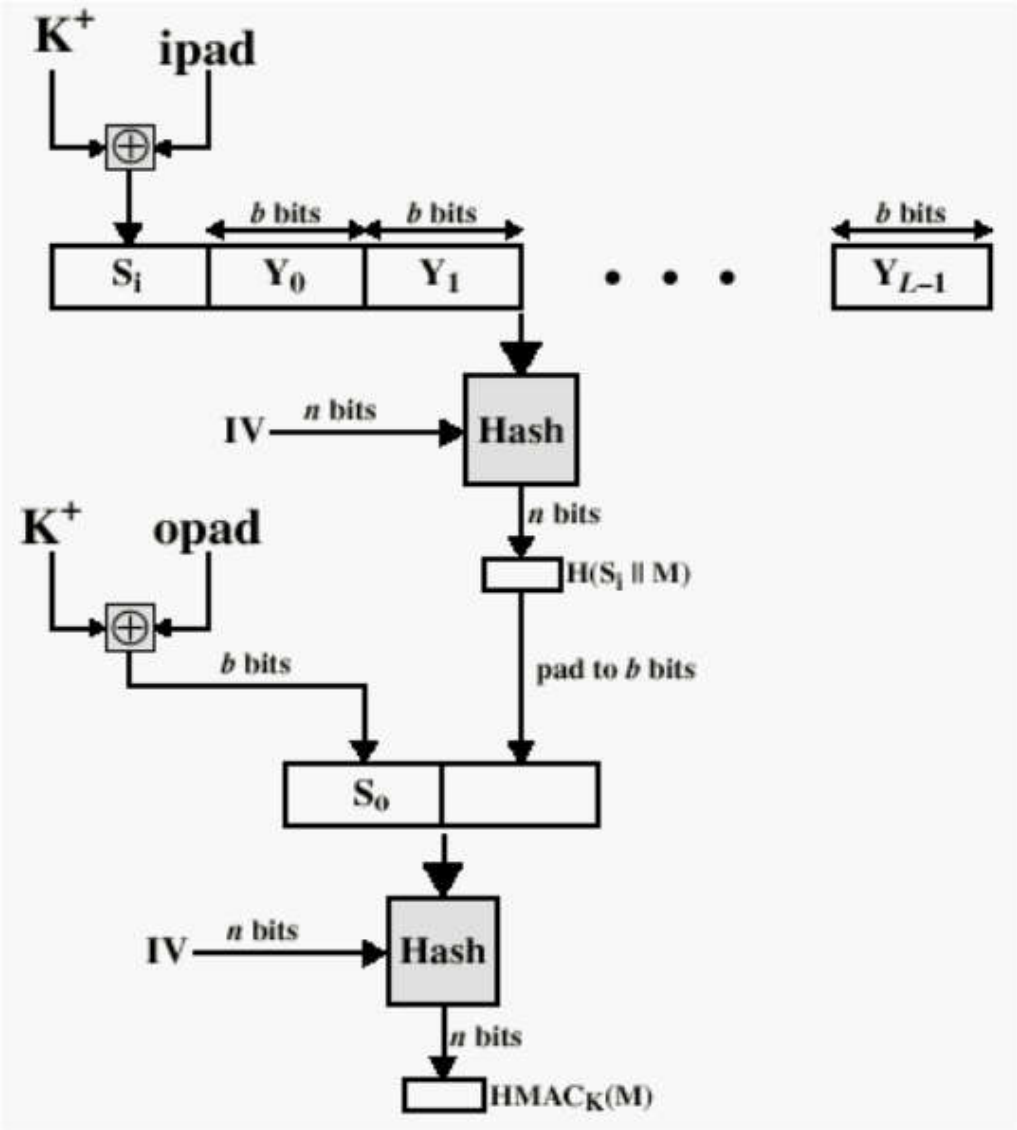
Secret prefix Klíč je přidán na začátek zprávy. Formálně: $H = h(k||x)$, kde H je výsledný hash (MAC), h je hashovací funkce, k je klíč a x je zpráva. Útočník může libovolně „přihashovávat“ další bloky bez znalosti klíče a tím vytvářet nové validní hashe – $h(k||x||y)$, kde y je útočníkova zpráva $\Rightarrow$ nepřijatelný způsob.

Secret suffix Klíč je přidán na konec zprávy. Formálně: $H = h(x||k)$. Útočník, který může zvolit x, může také vytvořit x', pro které $h(x) = h(x')$ se složitostí $O(2^{n/2})$, kde n je délka hashe, bez ohledu na délku klíče k (narozeninový útok) $\Rightarrow$ nepřijatelný způsob (útok který je neovlivnitelný délkou klíče).

Enveloping Klíč je přidán na začátek i na konec zprávy. Formálně: $H = h(k||p||x||k)$, kde p je zarovnání. Přijatelný způsob. Základ pro algoritmus HMAC.

9.5.3 HMAC (*hash function MAC*)

HMAC (*hash function MAC*) je do dnes používaný algoritmus. Specifikuje použití metody *enveloping*, ale ne, která hashovací funkce se použije.



Obrázek 9.6: Schéma HMAC; `ipad` a `opad` jsou vstupní/výstupní konstanty, které slouží k zarovnání; Y_i jsou bloky vstupní zprávy; IV je inicializační vektor.

Kapitola 10

KRY – Správa klíčů v asymetrické kryptografii (certifikáty X.509).

10.1 Metadata

- Předmět: Kryptografie (KRY)
- Přednáška:
  - `KRY04_Asym_MNG.pdf`
- Záznam:
  - 2021-03-29

10.2 Úvod a kontext

Viz. „Úvod a kontext“ v předchozích otázkách z tohoto předmětu.

Problém se zveřejňováním veřejných klíčů Jak můžu vědět, že publikovaný veřejný klíč patří opravdu entitě, které patřit má? Je potřeba zajistit autenticitu (pravost) veřejných klíčů – Vytvořit spolehlivou vazbu mezi veřejným klíčem a jménem jeho vlastníka.

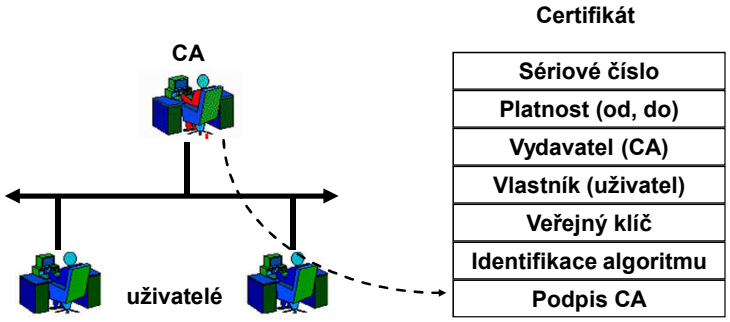
Systémy založené na veřejném klíči Systémy založené na veřejném klíči (PKI, *Public Key Infrastructure*) je označení infrastruktury správy a distribuce veřejných klíčů. PKI umožňuje pomocí přenosu důvěry používat cizí veřejné klíče a ověřovat jimi elektronické podpisy bez nutnosti jejich individuální kontroly.

10.3 Správa klíčů v asymetrické kryptografii

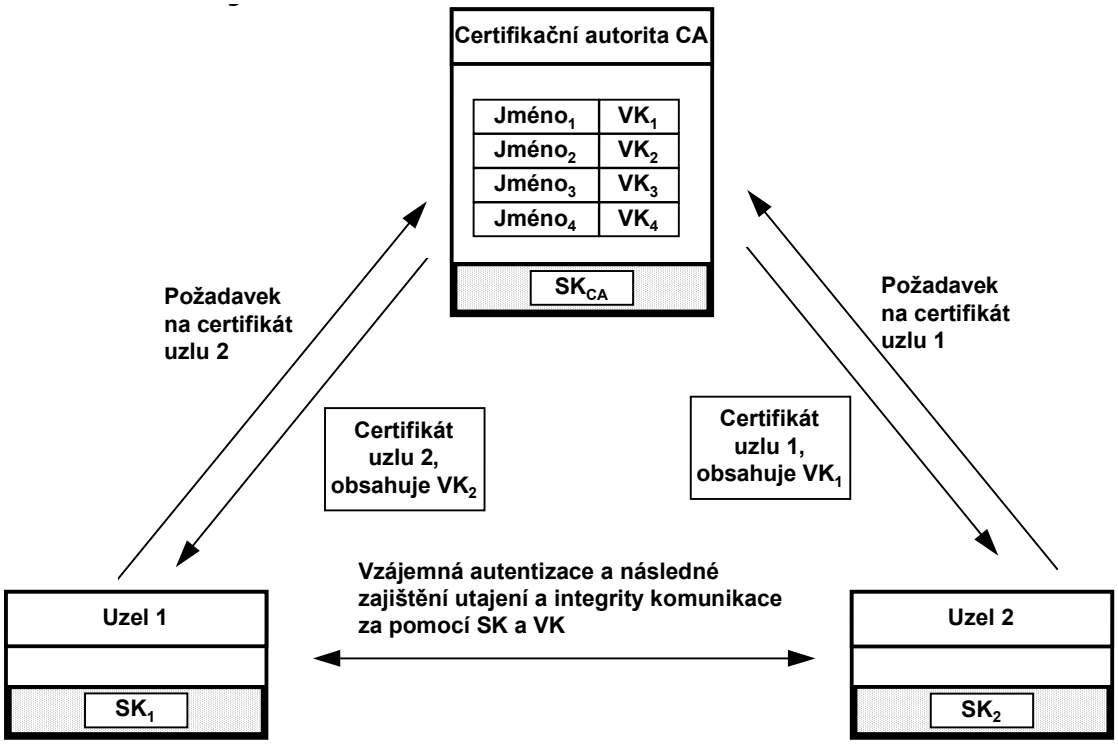
Certifikát Certifikace veřejného klíče. Nějaký prostředník (certifikační autorita), kterému důvěřujeme, se zaručuje, že konkrétní veřejný klíč, patří dané entitě.

Certifikační autorita Certifikační autorita (CA) je prostředník, který distribuuje certifikáty a které všichni důvěřují. CA negeneruje klíče uživatelům, ty si je generují samy.

Proces certifikace klíče CA podepíše veřejný klíč uživatele a jeho další údaje (jméno, doba vydání, doba platnosti, ...) svým soukromým klíčem. Tyto podepsané údaje se nazývají certifikát.



Obrázek 10.1: Příklad certifikátu.



Obrázek 10.2: Příklad navázání bezpečné komunikace mezi dvěma entitami, které mají stejnou certifikační autoritu.

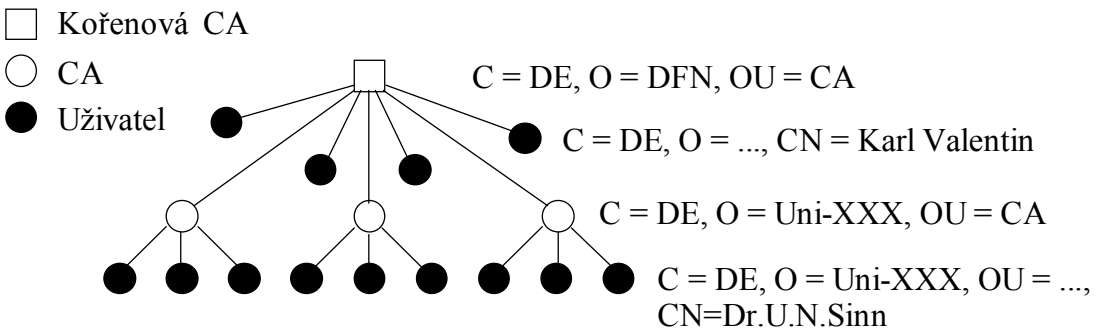
Navázání bezpečné komunikace Popis navázání bezpečné komunikace (viz obrázek 10.2):

1. Uzel 1 si vygeneruje soukromý a veřejný klíč.
2. Uzel 1 odešle veřejný klíč certifikační autoritě spolu se svým jménem (a dalšíma informacema).
3. CA vytvoří certifikát pro uzel 1 – svým soukromým klíčem podepíše veřejný klíč a jméno uzlu 1. CA odešle certifikát uzlu 1. CA odešle svůj veřejný klíč uzlu 1.
4. Pokud uzel 2 chce také odesílat, provede také kroky 1-3.

5. Uzel 1 podepíše soubor a odešle ho uzlu 2 (soubor a podpis).
6. Uzel 2 si musí sehnat certifikát uzlu 1. Existují 3 způsoby jak to udělat.
   - Odesílatel zašle svůj certifikát společně se zprávou.
   - Příjemce si vyžádá certifikát odesílatele od certifikační autority.
   - Příjemce si vyžádá certifikát odesílatele od jiné služby (adresářové služby, LDAP).
7. Uzel 2 ověří podpis u certifikátu uzlu 1 veřejným klíčem certifikační autority.
8. Uzel 2 ověří podpis souboru pomocí veřejného klíče odesílatele (který je v certifikátu).

Strom certifikačních autorit Model s jednou globální CA je nemožný (příliš mnoho uživatelů, přiliš velké vzdálosti, ...). Proto se používá strom certifikačních autorit. Veřejný klíč CA je certifikován jinou CA. CA nejvýše ve stromu se nazývá **kořenová certifikační autorita**.

- Certifikační autorita má svůj vlastní certifikát, který je podepsaný její certifikační autoritou.
- Koncový uživatel důvěřuje stále pouze jedné entitě – kořenové certifikační autoritě, ale přibývá jedna úroveň ověřování navíc.
- Příjemce dostane zprávu s podpisem. Musí znát certifikát odesílatele (podepsaný CA), certifikát certifikační autority (podepsaný CA_{root}) a veřejný klíč kořenové CA[1].
- Úrovní certifikačních autorit může být více (nejčastěji 1-2).



Obrázek 10.3: Příklad stromu certifikačních autorit. C, O, OU je identifikátor entity.

Certifikační cesta Posloupnost certifikátů od certifikátu kořenové CA přes certifikáty dalších CA až k certifikátu komunikující protistrany.

Zneplatnění certifikátu Jak zrušit platnost certifikátu? Normálně se zruší sám, až skončí jeho platnost. Pokud je potřeba certifikát zneplatnit před jeho vypršením je třeba využít tzv. revokační seznam (CRL, *certificate revocation list*). Důvody zneplatnění certifikátu:

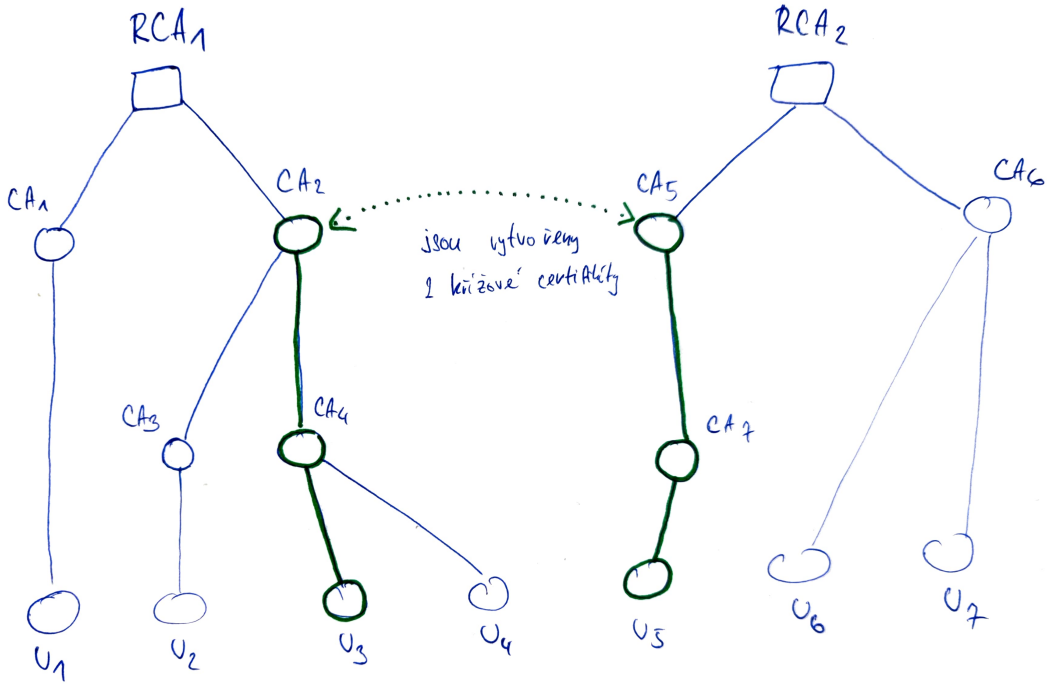
- soukromý klíč uživatele byl kompromitován,
- uživatel ztratil práva, která z certifikátu vyplývají (např. změna zaměstnavatele),
- soukromý klíč CA byl kompromitován (nikdy se nestalo).

[1] Veřejný klíč kořenové certifikační autority se z praktických distribuuje ve formě „fiktivního certifikátu“

CRL CRL (*certificate revocation list*) je seznam zneplatněných certifikátů, takových, kterým ještě nevypršela platnost, ale je třeba je zneplatnit. CRL je podepsán CA, které ho spravuje a periodicky aktualizuje (může se zkracovat i růst). Jak se distribuuje:

- *Pull model* – Příjemce certifikátu si dle potřeby stáhne CRL od CA.
- *Push model* – CA pravidelně posílá CRL příjemcům certifikátu.

Křížový certifikát V případě, že spolu komunikují uživatelé, kteří nemají společnou kořenovou CA (jejich CA si nedůvěřují), je třeba využít tzv. křížový certifikát. Tím se komplikuje sestavování certifikační cesty, protože je třeba zvážit všechny křížové certifikáty, které jsou k dispozici.



Obrázek 10.4: Příklad křížového certifikátu. Uživatel U_3 chce navázat bezpečné spojení s uživatelem U_5. Nemají společnou kořenovou CA, proto je třeba využít křížové certifikáty. CA2 vytvoří křížový certifikát pro CA_5 a CA_5 vytvoří křížový certifikát pro CA_2. Příklad: U_3 pošle podepsanou zprávu U_5, jak bude vypadat certifikační cesta? $U_3 \leftarrow CA_4 \leftarrow CA_2 \leftarrow CA_5 \leftarrow RCA_2$.

10.4 Standard X.509

X.509 je standard pro systémy založené na veřejném klíči (PKI). Specifikuje formát certifikátů, formát CRL, parametry certifikátů, metody kontroly platností certifikátů, ...

```
Certificate ::= SIGNED SEQUENCE {
    version [0] Version DEFAULT v1988,
    serialNumber CertificateSerialNumber,
    signature
    AlgorithmIdentifier,
    issuer
    Name,
    validity
    Validity,
    subject
    Name,
    subjectPublicKeyInfo SubjectPublicKeyInfo
}

Version ::= INTEGER {v1988(0) }

CertificateSerialNumber ::= INTEGER

Validity ::= SEQUENCE {notBefore UTCTime, notAfter UTCTime }

SubjectPublicKeyInfo ::= SEQUENCE {
    algorithm
    AlgorithmIdentifier,
    subjectPublicKey
    BIT STRING
}

AlgorithmIdentifier ::= SEQUENCE {
    algorithm
    OBJECT IDENTIFIER,
    parameters
    ANY DEFINED BY algorithm OPTIONAL
}
```

Výpis 10.1: Příklad definice certifikátu ve formátu X.509.

Význam položek Význam položek v definici certifikátu ve formátu X.509:

- Version – Standardně 0.
- Serial number – Sériové číslo certifikátu, spolu se jménem vydavatele jednoznačně identifikuje certifikát.
- Issuer – Jméno vydávající CA.
- Subject – Jméno vlastníka certifikátu.
- Validity – Doba platnosti certifikátu (`notBefore`, `notAfter`). Podpis je platný pouze pokud je datum podepsání v intervalu platnosti každého z certifikátů z certifikační cesty.
- SubjectPublicKeyInfo – Veřejný klíč vlastníka certifikátu a algoritmus, pro který je určen.
- Signature – Jakým algoritmem je certifikát podepsaný CA.

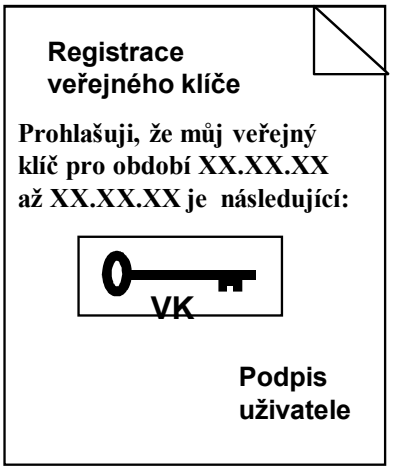
Prototypový certifikát Má strukturu certifikátu X.509. Uživatel si vygeneruje tzv. prototypový certifikát, který má standardní strukturu a vyplní informace na nějaké implicitní

hodnoty. Prototyp pošle CA spolu se svým veřejným klíčem, která certifikát dovyplní, podepíše a pošle zpět.

Registrační autorita Pokud chce uživatel vydat certifikát, kontaktuje tzv. registrační autoritu (nikoliv přímo CA).

Míra důvery v certifikát V praxi chceme více urovní důvěry, než pouze ostrý/žádný (např. chceme vytvořit testovací certifikát). To je řešeno jako rozšíření X.509 přidáním třídy certifikátu (*certification class*). Uživatel chce vydat certifikát od CA jisté třídy.

- Třída 1 – CA vůbec nekontroluje identitu žadatele. Lze jej získat anonymně. Používá se pro testovací certifikáty.
- Třída 2 – Identita žadatele musí být ověřena třetí stranou (notářsky ověřený formulář zaslaný poštou).
- Třída 3 – Standardní certifikát. Žadatel musí osobně navštívit CA (resp. registrační autoritu). Osobní ověření totožnosti.
- Třída 4 – Stejné jako 3 a navíc je nutné prokázat oprávnění žadatele požadovat certifikát.



Obrázek 10.5: Příklad žádosti o certifikát.

Kapitola 11

PDS – Základní architektury přepínačů, algoritmy pro plánování, řešení blokování, vícestupňové přepínací sítě.

11.1 Metadata

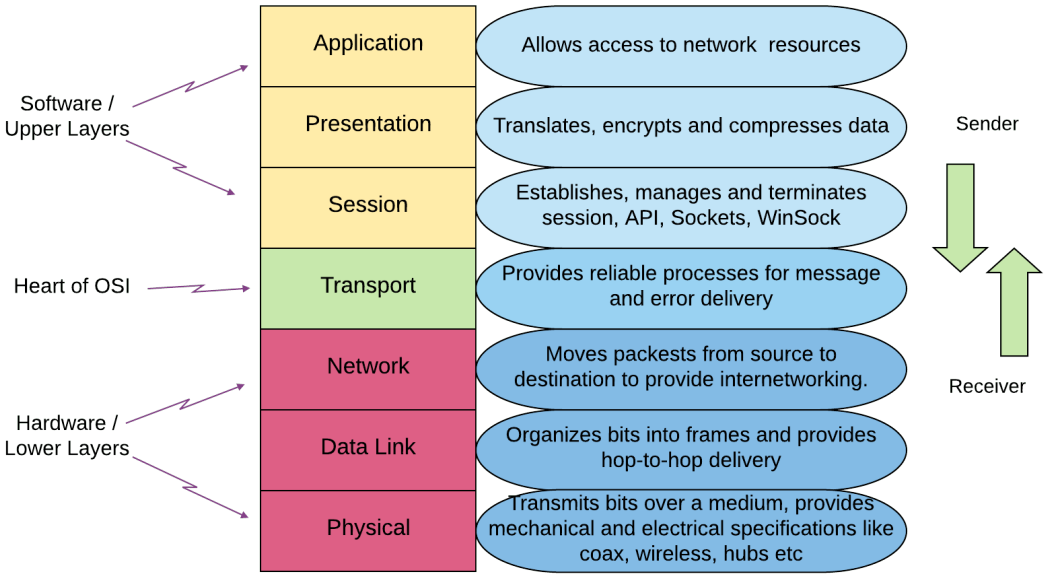
- Předmět: Přenos dat, počítačové sítě a protokoly (PDS)
- Přednáška:
  - `04-switching.pdf`
- Záznam:
  - 2021-03-05

11.2 Úvod a kontext

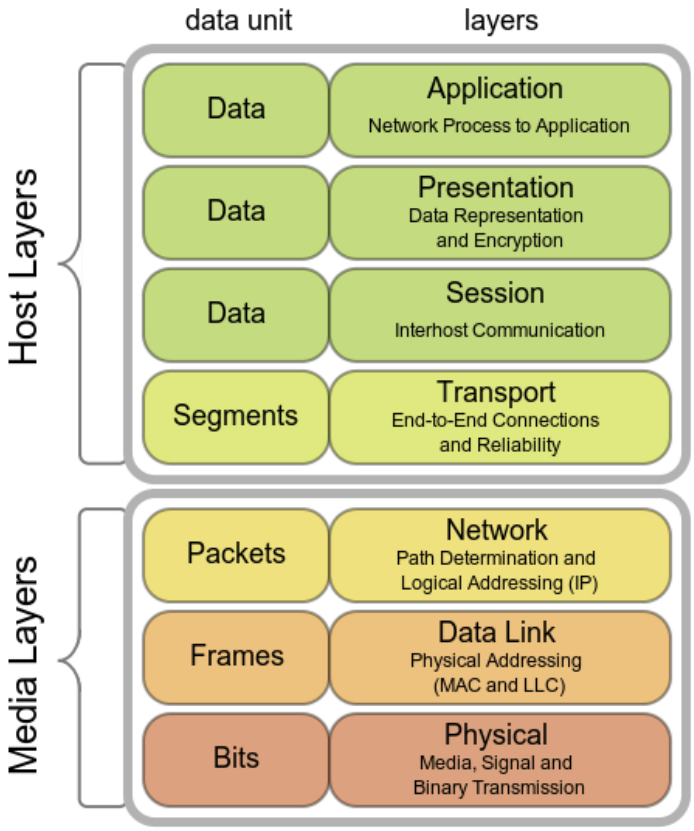
ISO/OSI model Referenční model ISO/OSI se používá jako názorný příklad řešení komunikace v počítačových sítích pomocí vrstevnatého modelu, kde jsou jednotlivé vrstvy nezávislé a snadno nahraditelné.

- **Aplikační vrstva** (L7, *application layer*)
  - Zajišťuje zpracování dat na nejvyšší úrovni (reprezentace dat, kódování, řízení dialogu, ...).
  - Tvořena procesy a aplikacemi, které komunikují po síti.
  - Bývá slučována s prezentační vrstvou (L6, prezentace dat a šifrování) a relační vrstvou (L5, koordinace a komunikace).
  - Příklad protokolů:
    * Uživatelské – vykonávají služby přímo uživateli (Telnet, SSH, FTP, SMTP, HTTP, ...)
    * Systémové – zajišťují síťové funkce (DNS, DHCP, SNMP, BOOTP, ...)
- **Transportní vrstva** (L4, *transport layer*)

  - Rozděluje aplikační data (segmentace) na menší jednotky a zapouzdřuje je do segmentů (TCP) / datagramů (UDP).
  - Vytváří logické spojení mezi procesy (přenáší data konkrétní aplikace ze zdrojového zařízení do aplikace na cílovém zařízení).
  - Adresace: porty.
  - Příklad protokolů: TCP, UDP, DCCP, SCTP, MP-TCP, QUIC
- **Síťová vrstva** (L3, *network layer*)
  - Zapouzdřuje segmenty/datagramy do paketů.
  - Řeší směrování.
  - Adresace: IP adresa (logická adresa).
  - Příklad protokolů:
    * IPv4, IPv6
    * ARP (Address Resolution Protocol), RARP (Reverse ARP) – Slouží k překladu MAC adres na IP adresy a obráceně.
    * ICMP (Internet Control Message Protocol) – Slouží pro řízení toku a detekce nedosažitelných uzlů.
    * IGMP (Internet Group Management Protocol)
- **Linková vrstva** (L2, *data link layer*, vrstva síťového rozhraní, *network interface layer*)
  - Zapouzdřuje pakety do rámců.
  - Zajišťuje *hop-by-hop* doručení.
  - Adresace: MAC adresa (fyzická adresace).
  - Příklad protokolů: Ethernet, Token Ring, FDDI, X.25, Frame Relay
- **Fyzická vrstva** (L1, *physical layer*)
  - Zajišťuje přenos bitů přes fyzické médium.



Obrázek 11.1: Příklad OSI modelu z Linuxhint.



Obrázek 11.2: Příklad OSI modelu z Wiki.

Přepínač Síťový přepínač (*switch*) je aktivní prvek v počítačové síti, který propojuje jednotlivé prvky do hvězdicové topologie. Přepínač obsahuje síťové porty (až stovky), na

něž se připojují síťová zařízení.

- Na jaké vrstvě OSI modelu pracuje? Pracuje s rámci (linková vrstva, L2).
- Na základě čeho provádí přepínání? Na základě cílové MAC adresy.
- Jaké typy přenosů přepíná? Na základě cílové MAC adresy rozlišuje typ přenosu:
  - broadcast (samé 1),
  - multicast (speciální prefix pro IPv4 a IPv6),
  - unicast (cokoliv jiného).
- Co ovlivňuje rychlost přepínání?
  - Hardware (typ paměti, rychlost procesoru)
  - Logika přepínání
  - Šířka pásma cílového rozhraní

CAM tabulka CAM (*Content Addressable Memory*) tabulka je datová struktura v přepínači, která uchovává informace, jako jsou MAC adresy dostupné na fyzických portech s přidruženými parametry VLAN.

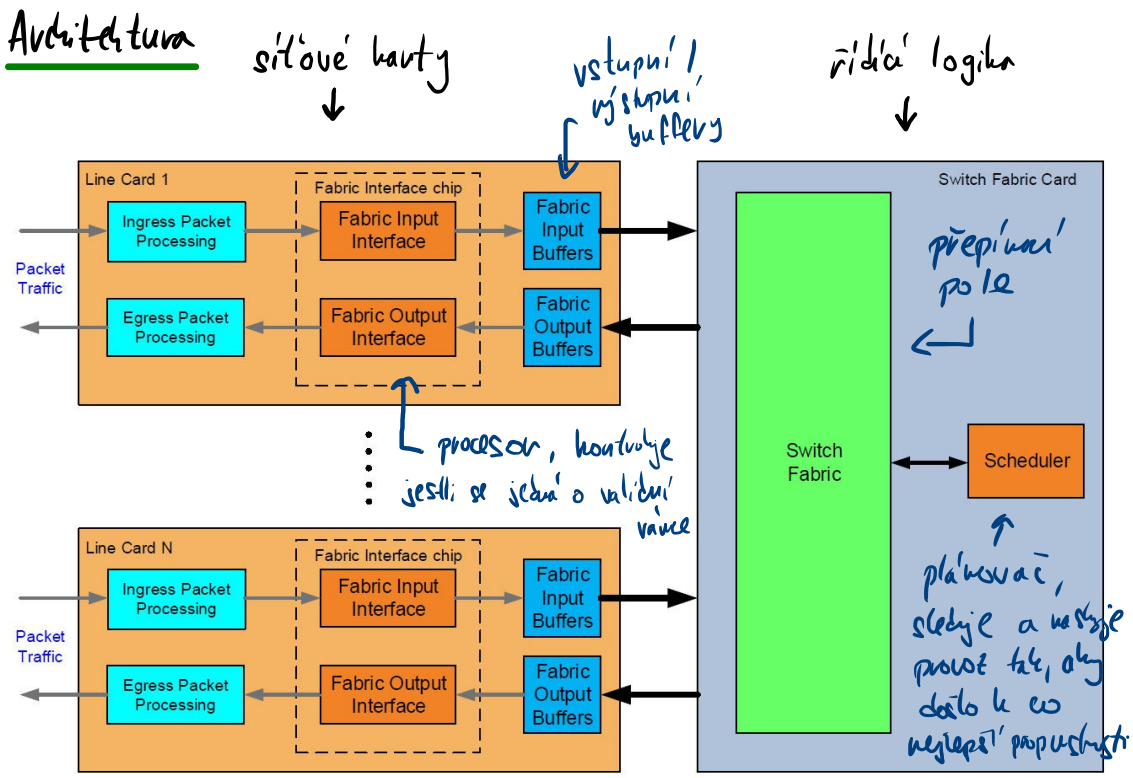
Propustnost Kolik je možné přenést dat za časovou jednotku.

11.3 Obecná architektura přepínače

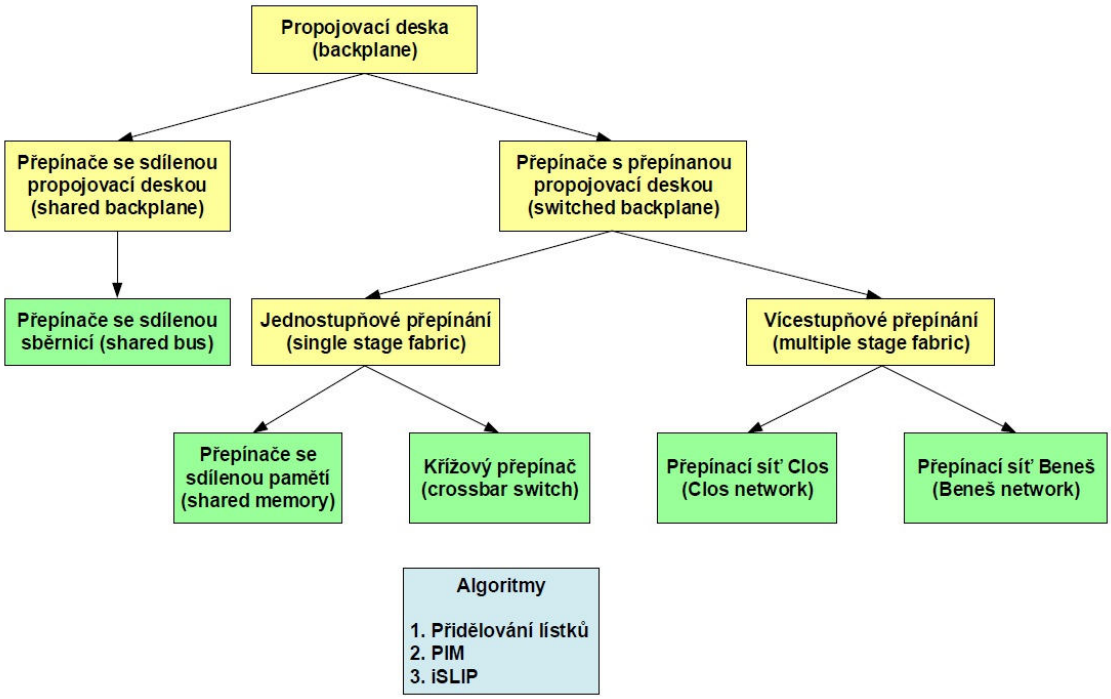
Viz obrázek 11.3.

Požadavky na přenos

- Maximální využití sběrnice (požadavek na plánovač).
  - Aby došlo k maximálnímu přenosu dat přepínací logikou (co nejvíce přenosů v rámci taktu – „paralelizace“).
- Spravedlivé přidělování přenosového pásma (aby byly obsluhovány všechny porty).
- Zachování pořadí rámců (ne vždy je možné).



Obrázek 11.3: Obecná architektura přepínače.



Obrázek 11.4: Dělení přepínačů.

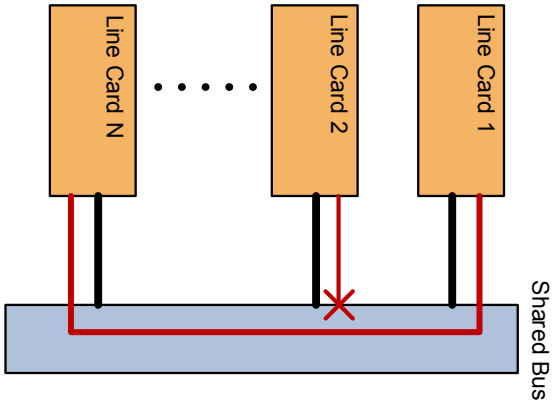
11.4 Přepínače se sdílenou propojovací deskou

11.4.1 Přepínače se sdílenou sběrnicí

- Může komunikovat pouze jeden port v daný čas, ostatní čekají (řízeno protokolem).
- Lze jednoduše realizovat broadcast a multicast.
- Mějme N portů (karet), rychlost každé karty R bps, taktovací frekvenci r. Pak rychlost sběrnice musí být $N \times R$ a šířka sběrnice

$$W = \frac{R \times N}{r}$$

- Pokud se přidají nové porty nebo se na nich zvýší rychlost je potřeba zvýšit propustnost sběrnice („něco za něco“).



Obrázek 11.5: Přepínač se sdílenou sběrnicí.

Úloha č.1:

- Přepínač má 16 portů o rychlosti 100 Mb/s. Jaká je potřebná propustnost sběrnice a jaká musí být šířka sběrnice, aby zvládl požadovaný přenos? Uvažujte taktovací frekvenci sběrnice 40 MHz.

$r = 40$ MHz $\quad N = 16 \quad R = 100$ Mb/s

propustnost – kolik dat za časovou jednotku

$\hookrightarrow N \times R = 1{,}6$ Gb/s

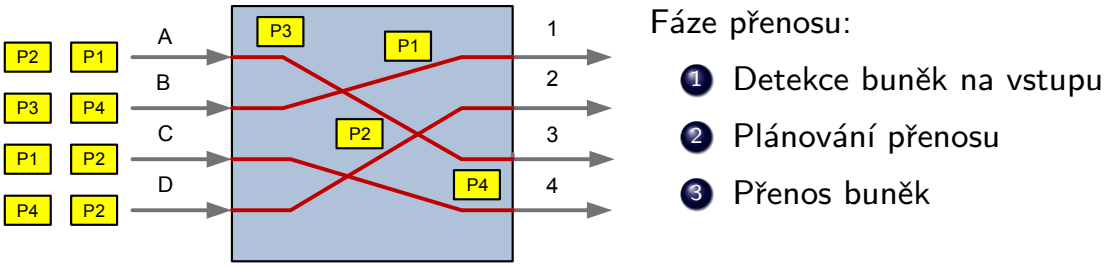
šířka sběrnice – kolik v jeden okamžik maximálně

$\hookrightarrow \frac{N \times R}{r} = \frac{16 \cdot 100 \cdot 10^6}{40 \cdot 10^6} = 40$ b

Obrázek 11.6: Přepínač se sdílenou sběrnicí – příklad.

11.5 Přepínače s přepínanou propojovací deskou: jednostupňové přepínání

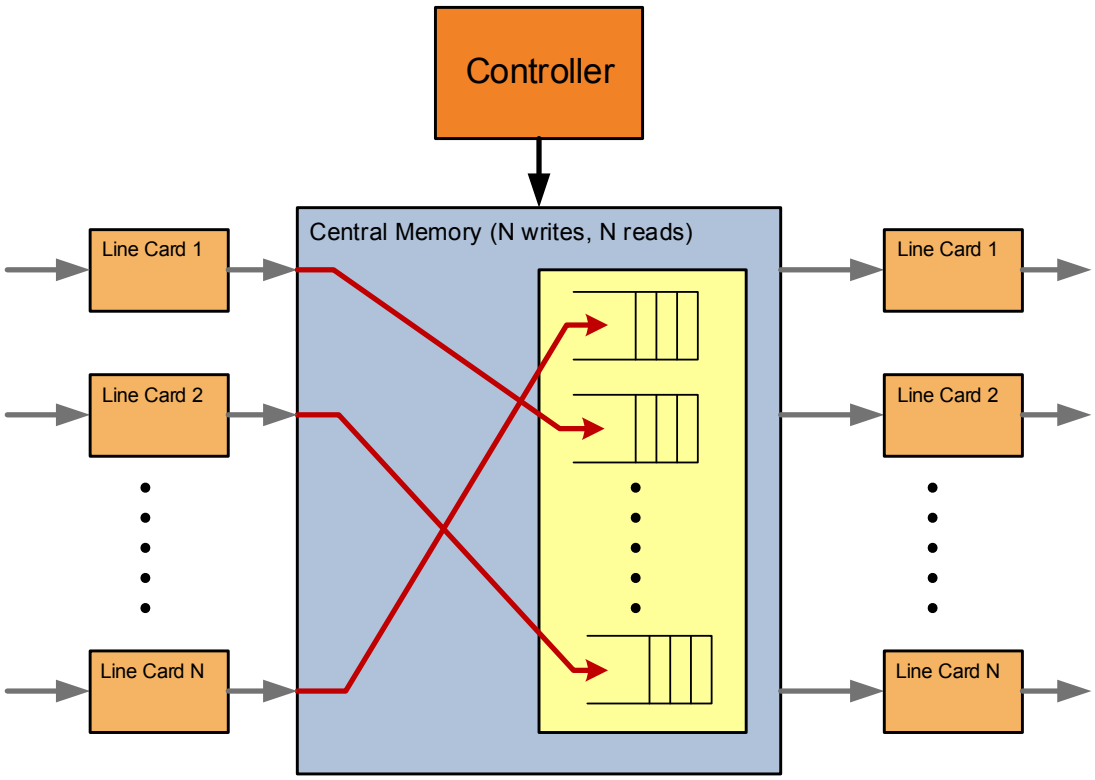
- Umožňuje paralelní přenos rámců.
- Je potřeba plánovač (*scheduler*), který plánuje přenos.
- Provedení přenosu: z bufferů vstupních karet se data přenesou do bufferů výstupních karet.



Obrázek 11.7: Činnost přepínače s přepínanou propojovací deskou.

11.5.1 Přepínače se sdílenou pamětí

- Centrální sdílená paměť, která obsahuje frontu pro každý výstupní buffer (každý port).
- Pro jeden přenos se musí 2x přistoupit do paměti (zápis a čtení).
- Zde narážíme na současné technologické limity pamětí (rychlost přístupu).



Obrázek 11.8: Činnost přepínače s přepínanou propojovací deskou a se sdílenou pamětí. Obsahuje: N vstupních karet, řadič, sdílenou paměť s N frontami, N výstupních karet.

- Nechť R je rychlost síťové kart, N je počet síťových karet, pak celková propustnost přepínače je $BW = 2 \times N \times R$ (dva přístupy do paměti).
- Mějme data o velikosti C, pak doba přenosu je $t = C \div BW[s]$.

Úloha č.2:

- Jaká je potřebná doba přístupu do paměti u přepínače se sdílenou pamětí, který má 32 portů o rychlosti 1 Gb/s a velikost ukládané buňky je 40 bytů? Jak se tato doba změní pro rychlosti portů 10 Gb/s a 40 Gb/s?

$C = 40\ B \qquad N = 32 \qquad R = 1\ Gb/s$

$$t = \frac{40 \cdot 8}{2 \cdot 32 \cdot 1 \cdot 10^9} = 5 \cdot 10^{-9}\ s = 5\ ns$$

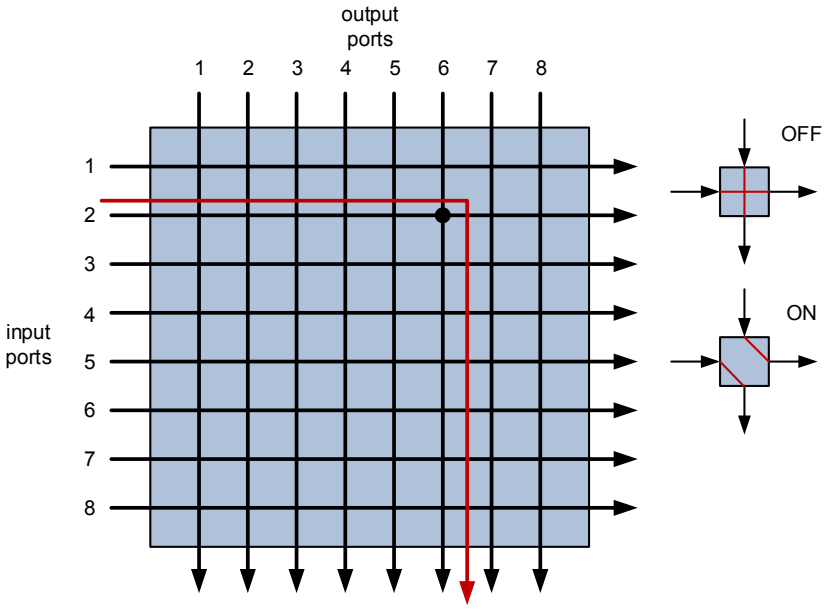
pro 10 -> 0,5 ns

pro 40 -> 0,125 ns

Obrázek 11.9: Přepínač s přepínanou sběrnicí a se sdílenou pamětí – příklad.

11.5.2 Křížové přepínače (*crossbar*)

- Založený na jednom velkém propojovacím poli.
- Interně neblokující.
- Nativní podpora multicastu.
- Je potřeba N^2 propojení.
- Chybí redundance: pouze jedna cesta mezi vstupem a výstupem.



Obrázek 11.10: Přepínač s přepínanou sběrnicí a křížovými propoji.

11.6 Hledání spojů v křížovém přepínači

11.6.1 Problém párování

V kontextu přepínání v křížovém přepínači se jedná o problém párování v bipartitním grafu, kde množina vstupních portů je V_1, množina výstupních portů je V_2 a hrany E jsou propoje vstupů a výstupů. Množina M se nazývá párování, pokud $M \subseteq E$, kde žádné dvě hrany z M, nemají společný vrchol.

Bipartitní graf Nechť $G = (V, E)$ je graf. Graf G je bipartitní, pokud platí $V = V_1 \cup V_2 \wedge V_1 \cap V_2 = \emptyset \wedge \forall e = \{u, v\}, e \in E : u \in V_1 \wedge v \in V_2$. Platí li navíc $E = V_1 \times V_2$, pak je graf úplný bipartitní.

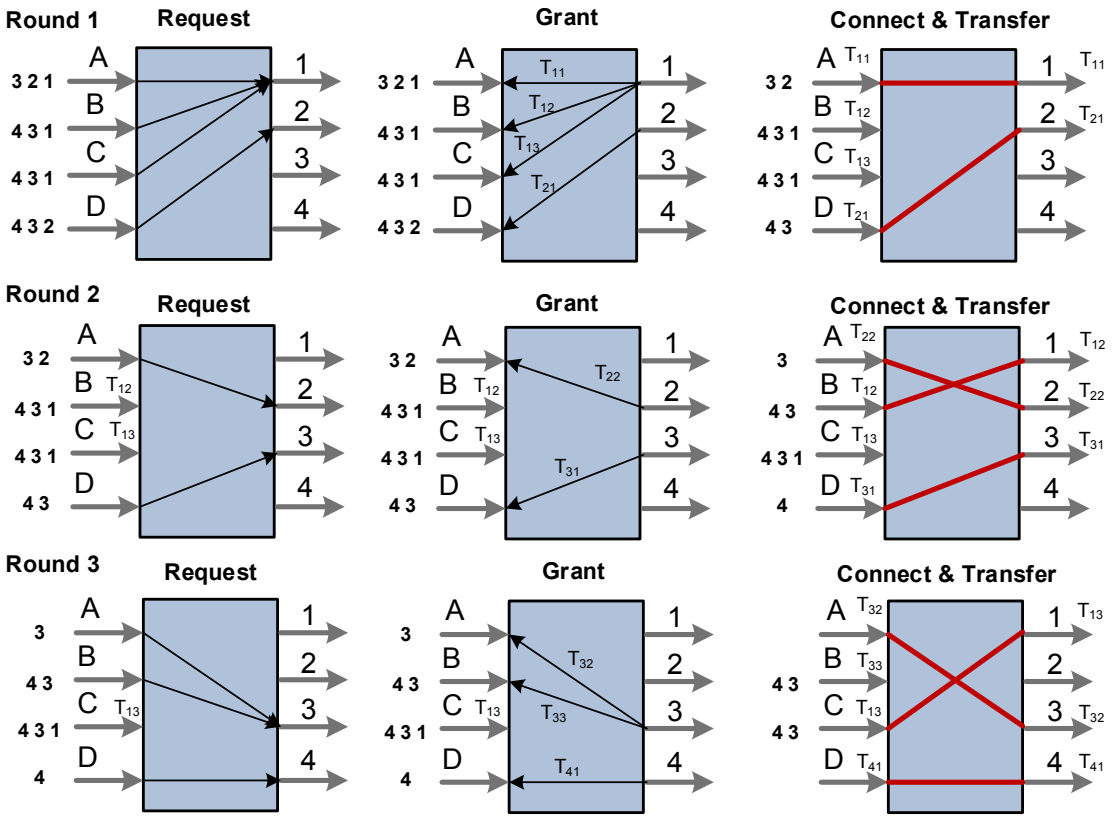
Největší párování Největší párování (*maximum matching*) je párování, které má největší počet hran (globální maximum). Složitost: $O(N^{5/2})$.

Maximální párování Maximální párování (*maximal matching*) je párování, kdy nelze přidat žádnou další hranu (lokální maximum). Složitost: $O(N + E)$.

Jelikož algoritmy pro hledání maximálního párování mají výrazně menší časovou složitost, volíme pro hledání spojů v křížovém přepínači ty.

11.6.2 Algoritmus přidělování lístků

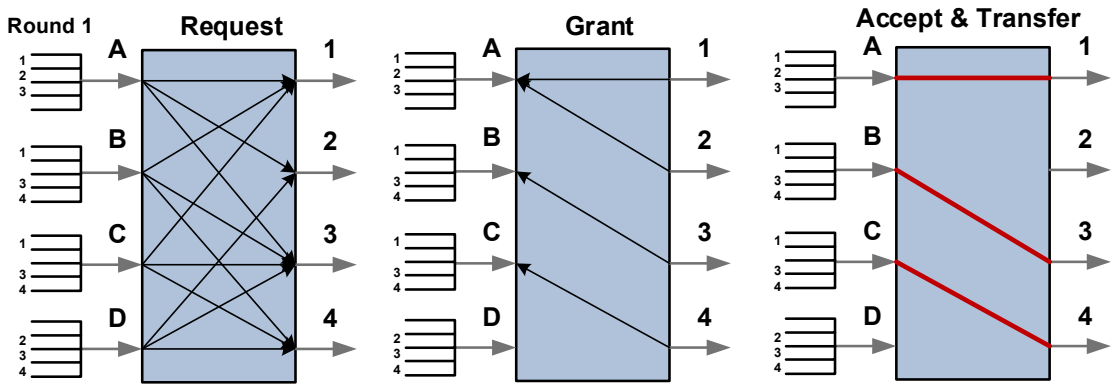
- Princip:
  - Výstupní port Q obsluhuje frontu požadavků na propojení.
  - Požadavek na vstupu P dostane od Q číslo T_{QX} pro obsloužení portu Q s pořadím X.
- Fáze:
  1. Žádost o lístek
  2. Přidělení lístku
  3. Propojení vstupů s výstupy a přenos
- Hodnocení:
  - Umožňuje přenos rámců s proměnlivou délkou
  - **Blokování na začátku fronty** (*head of line blocking*) – Situace, kdy více vstupů chce stejný výstup. Musí se zpracovat ve více kolech. Řešení: Virtuální výstupní fronty (každý vstupní port má frontu pro každý výstupní port). Tím může dojít k předbíhání rámců. Viz další algoritmy.



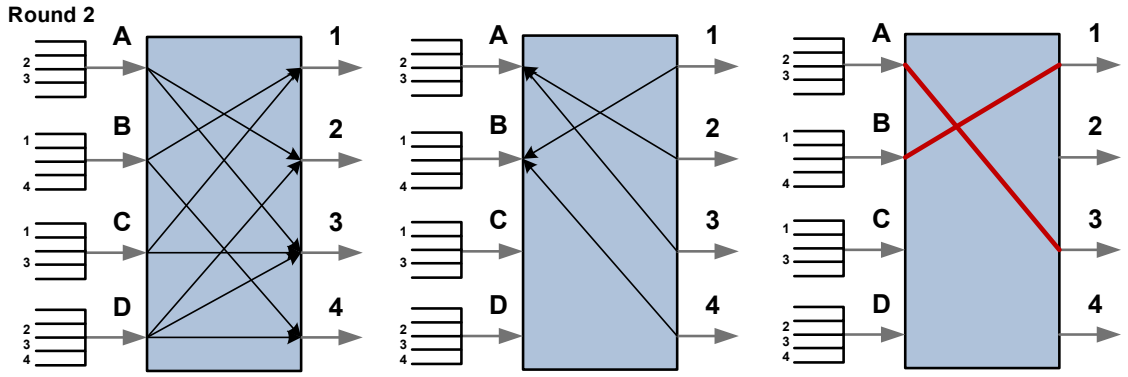
Obrázek 11.11: Příklad činnosti algoritmu přidělování lístků.

11.6.3 Algoritmus PIM (*Parallel Iterative Matching*)

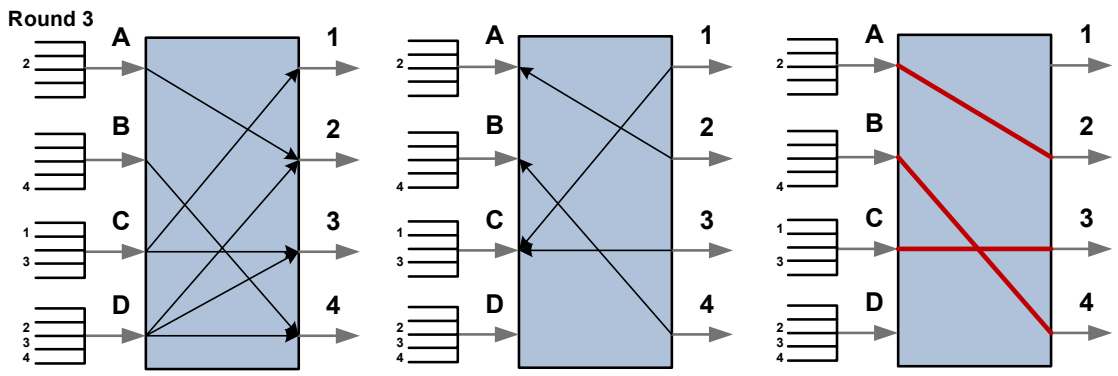
- Algoritmus přidělování lístků + virtuální výstupní fronty + náhodnost.
  - Posílají se požadavky pro všechny pakety ve frontě, nikoliv pouze pro první.
- Hledá maximální párování (pomocí náhodné volby, která zabrání vyhladovění).
- Soutěžení o porty
  - Výstupní port – více žádostí naráz, jednu vyberu náhodně.
  - Vstupní port – povolení na více výstupů naráz, jeden vyberu náhodně.



Obrázek 11.12: Příklad činnosti algoritmu PIM, kolo 1.



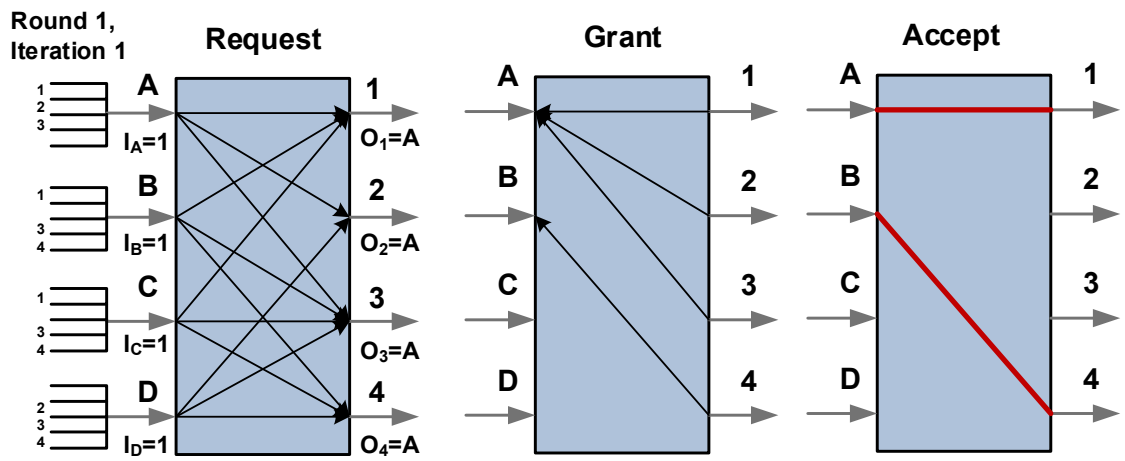
Obrázek 11.13: Příklad činnosti algoritmu PIM, kolo 2.



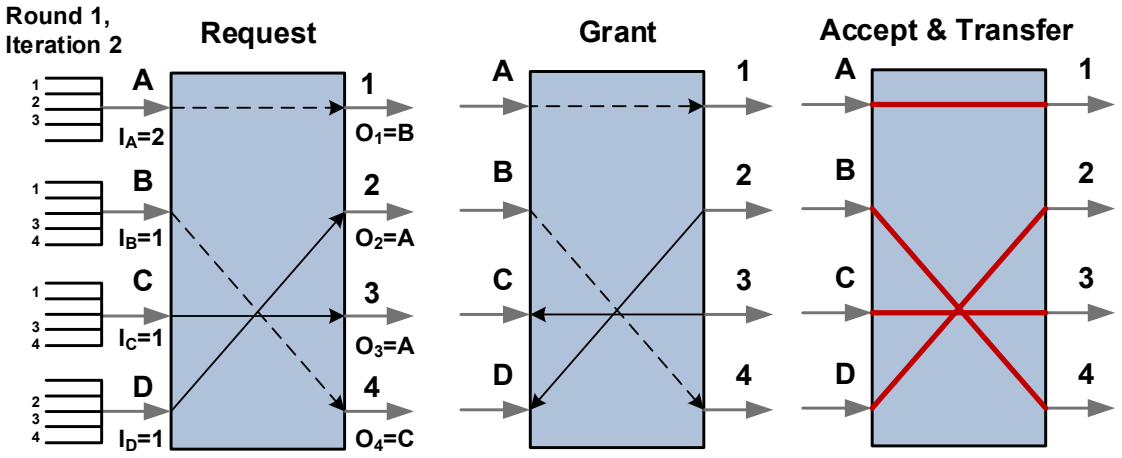
Obrázek 11.14: Příklad činnosti algoritmu PIM, kolo 3.

11.6.4 Algoritmus iSLIP

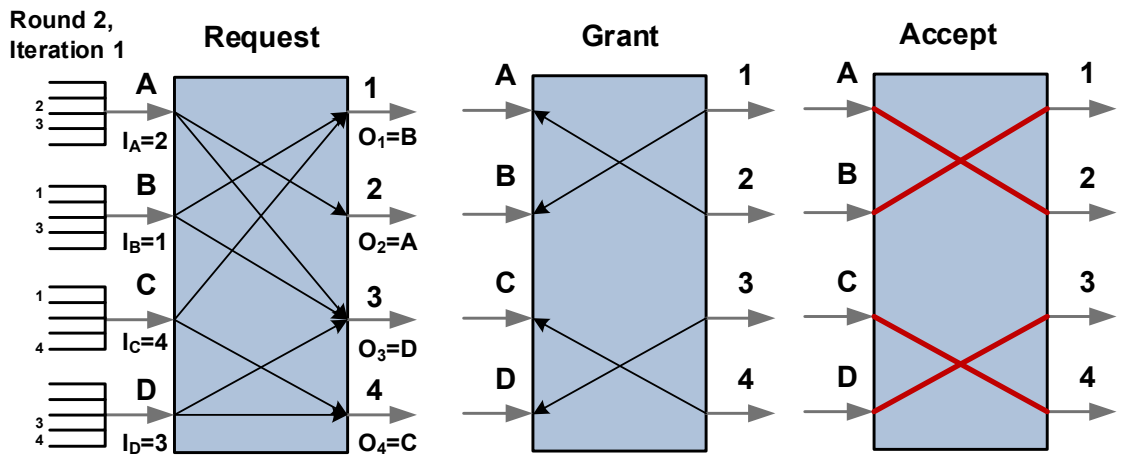
- Algoritmus přidělování lístků + virtuální výstupní fronty + iterace + ukazatele.
- Algoritmus „rotujících ukazatelů“ (vyhneme se náhodnému výběru).
  - Každý vstupní port má ukazatel (počítadlo), podle kterého se určuje priorita výstupu.
  - Každý výstupní port má ukazatel, podle kterého se určuje priorita vstupu.
  - Nová hodnota ukazatele je port na který se posílá / port od kterého se přijímá +1.
- Každé kolo má dvě iterace, resp. 6 fází – *request*, *grant*, *accept*, *request*, *grant*, *accept and transfer*.
  - Proč? Protože fáze *transfer* trvá nejdéle (optimalizace).



Obrázek 11.15: Příklad činnosti algoritmu iSLIP, kolo 1.



Obrázek 11.16: Příklad činnosti algoritmu iSLIP, kolo 2.



Obrázek 11.17: Příklad činnosti algoritmu iSLIP, kolo 3.

11.7 Přepínače s přepínanou propojovací deskou: vícestupňové přepínání

Vlastnosti jednostupňového křížového přepínače:

- S počtem portů roste kvadraticky velikost přepínacího pole.
- Vnitřní blokování (soupeření o porty).
- Blokování na začátku fronty (*head of line blocking*).
- Součástí přepínání je hledání maximálního párování.

Je možné zvětšit počet portů, aniž by se dramaticky rozšířila velikost přepínacího pole? Ano, pomocí vícestupňového přepínání (vstup → vnitřní přepínací obvody → výstup).

- Nemají vnitřní blokování – Nejsou potřeba plánovací algoritmy pro hledání maximálního párování.

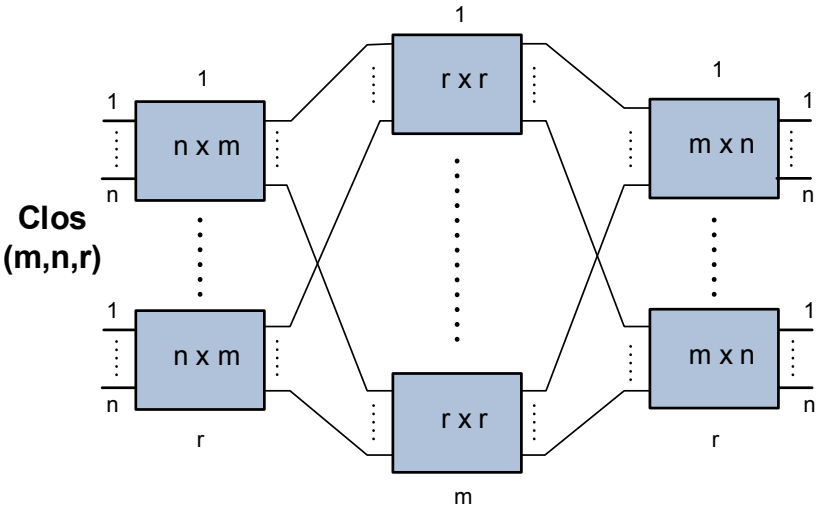
11.7.1 Přepínací síť Clos

$Clos(m, n, r)$

- r vstupních bloků s n vstupy (bloků = křížový přepínač)
- r výstupních bloků s n vstupy
- m vnitřních bloků s r vstupy

Closův teorém:

- Pokud $m \geq 2n - 1$, pak lze přidat nové propojení vstupu a výstupu bez přeskládání (síť je neblokující).
  - Pokud všechny požadavky na vstupu jdou na jiné výstupy, pak pro jakoukoliv konfiguraci vstupů a výstupů je síť neblokující (pokud nejdou dva vstupy na jeden stejný výstup).

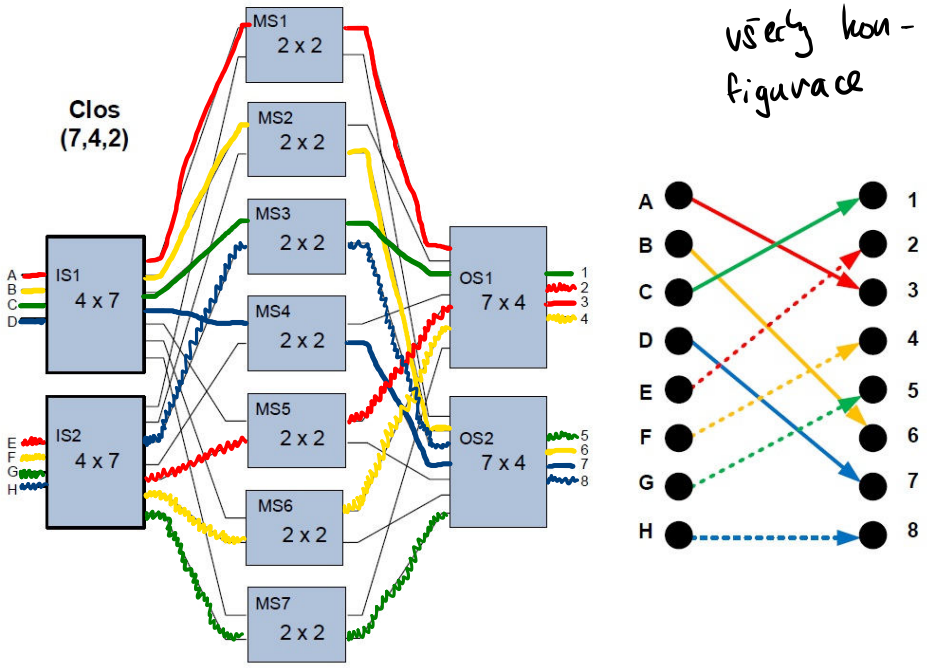


Obrázek 11.18: Obecné schéma Closovy sítě.

Příklad neblokující sítě Clos(7,4,2): osmiportový přepínač

- Platí Closův teorém, tj. $m \geq 2n - 1$

• Vždycky najde propojení přes všechny konfigurace



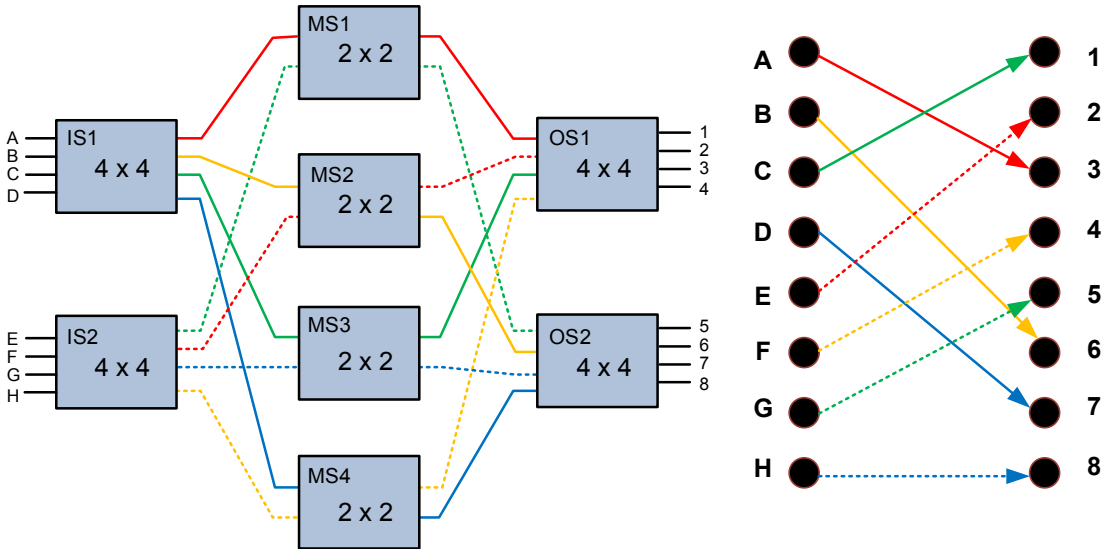
Obrázek 11.19: Příklad Closovy sítě.

11.7.2 Modifikovaná přepínací síť Clos

- Pokud $m \geq n$, pak je síť neblokující po přeskládání.
- Výpočetní složitost přeskládání je $O(N \log D)$, kde D je počet barev.
- Proč? Při plnění closova teorému je potřeba hodně bloků. Při této podmínce stačí výrazně méně a složitost výpočtu přeskládání je přijatelná.

Příklad sítě Clos(4,4,2): osmiportový přepínač

- Příklad neblokující konfigurace po přeskládání.

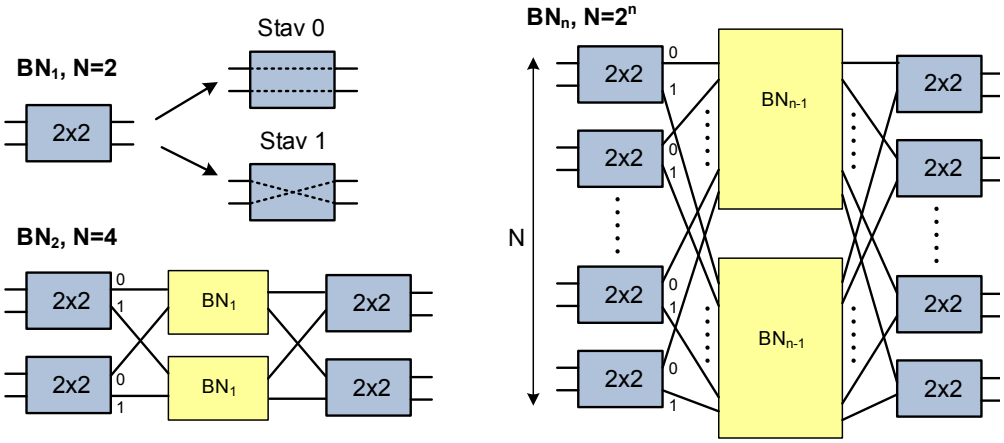


Obrázek 11.20: Příklad modifikované Closovy sítě.

11.7.3 Přepínací síť Beneš

$Benes(n)$ také $BN(n)$

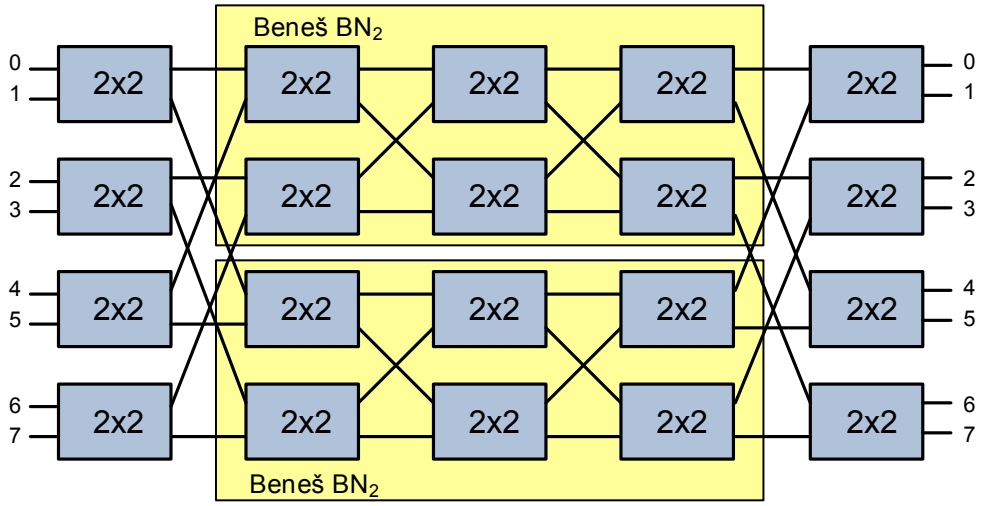
- Základem je modifikovaná síť $Clos(m = 2, n = 2, r = 1)$.
- Rekurzivní konstrukce sítě $Benes(n)$, kde n je stupeň rekurze.
- Vstupní a výstupní přepínače velikosti 2×2.
- Celkový počet vstupních/výstupních portů $N = 2^n$.
- Prostřední část rekurzivních bloků $BN(n-1)$.
- Neblokující po přeskládání – algoritmus hledání propojení (žádné soupeření o vnitřní uzly nebo výstupní porty).



Obrázek 11.21: Obecné schéma Benešovy sítě.

Příklad: přepínací síť Beneš BN_3

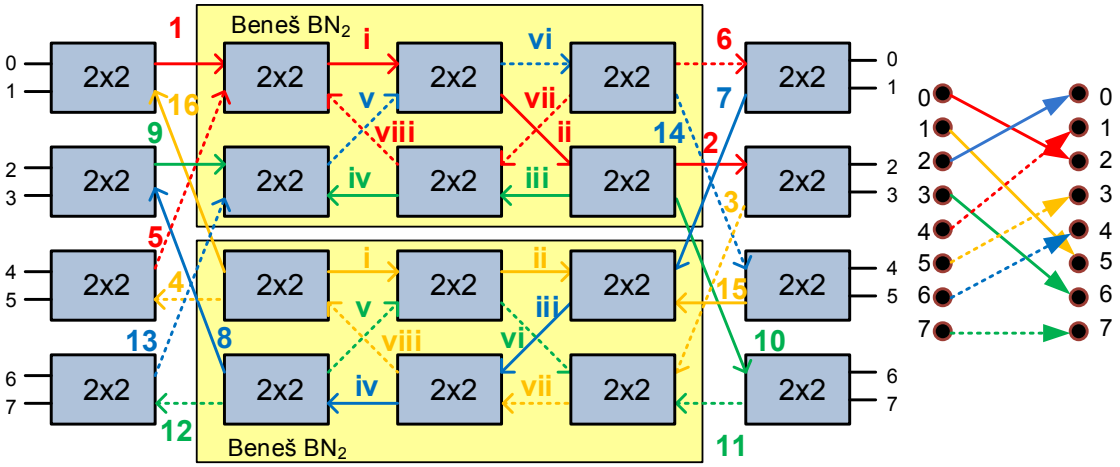
- N = 8 vstupů, 5 stupňů
- Použito např. u směrovače Cisco CSR-1.



Obrázek 11.22: Příklad Benešovy sítě.

Algoritmus hledání propojení

- Horní podsíť BN_i slouží pro dopředné směrování, spodní podsíť BN_i pro zpětné.
- Postupuje se od vstupů k výstupům a zpět, vždy pro sousední porty.
- Nejprve se propojí vstupní a výstupní bloky, poté rekurzivně síť BN_{i-1}.



Obrázek 11.23: Příklad Benešovy sítě s přeskládáním.

Kapitola 12

PDS – Základní funkce směrovače, zpracování paketů ve směrovači, typy přepínání a architektur.

12.1 Metadata

- Předmět: Přenos dat, počítačové sítě a protokoly (PDS)
- Přednáška:
  - `05-routing.pdf`
- Záznam:
  - 2021-03-12

12.2 Úvod a kontext

[[**todo**]]

Kapitola 13

PDS – Řízení toku dat (flow-control) a prevence zahlcení (congestion-control) na transportní vrstvě (MP-TCP, QUIC, SCTP, DCCP).

13.1 Metadata

- Předmět: Přenos dat, počítačové sítě a protokoly (PDS)
- Přednáška:
  - `02-transportni-protokoly.pdf`
- Záznam:
  - 2021-02-19

13.2 Úvod a kontext

Viz. „Úvod a kontext“ v předchozích otázkách z tohoto předmětu.

Transportní vrstva Transportní vrstva je název čtvrté vrstvy modelu vrstvové síťové architektury (ISO/OSI model). Leží mezi vrstvou síťovou (L3) a aplikační (L7).

- Činnost na straně odesílatele: obdrží data z aplikační vrstvy, nasegmentuje je a ke každému segmentu/datagramu přidá L4 hlavičku.
- Činnost na straně příjemce: obdrží segmenty/datagramy, které uspořádá do správného pořadí (*out-of-order* doručení), předá je aplikační vrstvě.
- Komunikace mezi vrstvami L4 a L7 probíhá pomocí socketů.
- Zodpovědnost za *end-to-end* spojení.
- Adresuje aplikace pomocí portových čísel.
- *Quality of Service* – zotavení se z chyb, spolehlivost, řízení toku, řízení zahlcení.

Multi homing – Využití více bodů připojení zároveň (pro jednu komunikaci). Pokud je podporováno a změní se IP adresa, nemusí se navázat nové spojení.

Connection-oriented – Typ spojení, kdy je nejprve nutné, navázat komunikaci. Typicky tři fáze: navázání spojení, přenos dat, ukončení spojení.

Connection-less – Typ spojení, které nerozlišuje jestli spojení existuje, nebo nikoliv. Pouze fáze přenosu dat.

Maximum transmission unit (MTU) Největší velikost paketu, kterou lze v síti odeslat.

Fragmentace paketů Uzel v síti (směrovač) dostane paket o velikosti n. Paket má přeposlat do sítě, ve které je $MTU < n$. Aby paket mohl být odeslán, musí být rozdělen (fragmentován) na více menších paketů (fragmenty) a odeslán po částech. Na straně příjemce pak musí nastat opačný proces – defragmentace.

Round trip time (RTT) RTT je doba, která uplyne od vyslání paketu z jednoho komunikujícího uzlu k druhému po návrat potvrzení zpátky na první uzel.

bandwith, throughput, goodput

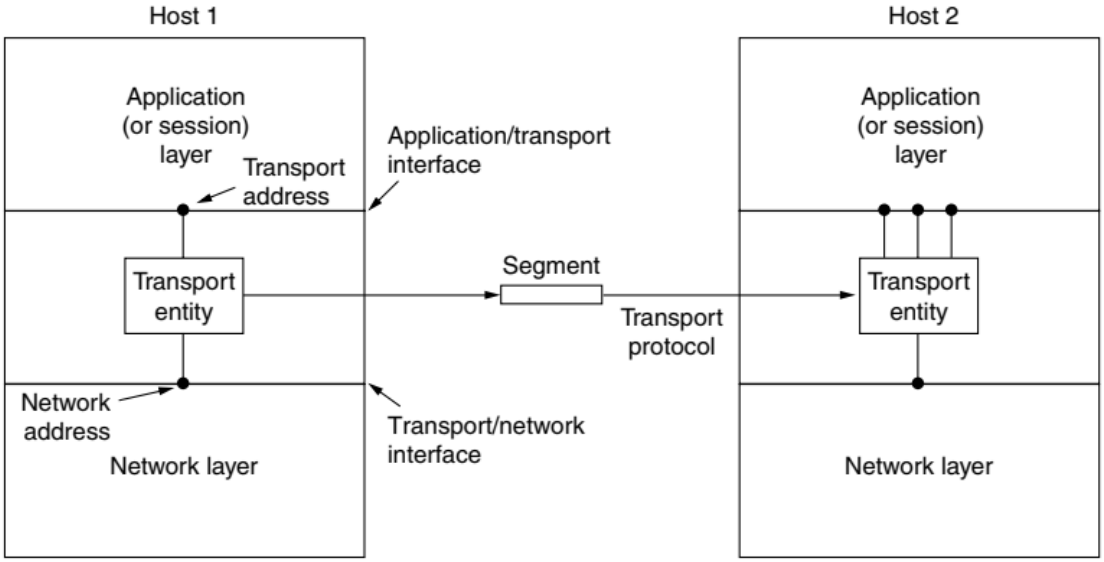
- *Bandwith* – šířka pásma (maximální přenos dat danou cestou v jeden moment)
- *Throughput* – propustnost komunikace na úrovni protokolu
- *Goodput* – propustnost komunikace na úrovni aplikace (throughput − režie)

data, segment, datagram, paket, rámec, bit

- Data – aplikační vrstva (L7)
- Segment – transportní vrstva (L4), TCP
- Datagram – transportní vrstva (L4), UDP
- Paket – síťová vrstva (L3)
- Rámec – linková vrstva (L2)
- Bit – fyzická vrstva (L1)

Bitové chyby

- „Přeskočení bitu“ (*bit swapping*), jde o chyby na úrovni zpracování informace uzlem v síti (typicky spojené s přenosovým médiem)
- Zabýváme se jimi na úrovni L2
- Řešením je obvykle redundance (Hammingův kód, CRC, Viterbiho algoritmus, ...)



Obrázek 13.1: Transportní vrstva (L4).

13.3 Paketové chyby

- Jde o chyby na úrovni přenosu informace mezi dvěma uzly.
- Zabýváme se jimi na transportní vrstvě.
- Řešením je obvykle znovuzaslání paketu.
- Můžeme měřit:
  - PER (*packet error rate*)

$$PER = \frac{\text{pocet chybnych paketu}}{\text{pocet prenesenych paketu}}$$

  - BER (*bit error rate*)

$$BER = \frac{\text{pocet chybnych bitu}}{\text{pocet prenesenych bitu}}$$

- Dále jsou představeny druhy paketových chyb.

Ztráta (zpoždění) paketu Jak může dojít ke ztrátě (zpoždění) paketu?

- Neopravitelná bitová chyba (rámec je zahozen na L2)
- Zahlcení linky (směrovače jsou přetíženy a některé pakety zahazují)
- Špatné směrovací tabulky (paket jde špatnou cestou, nebo cyklí a vyprší mu TTL)

Ztráta fragmentovaných dat Fragment se může ztratit ze stejného důvodu jako standardní nefragmentovaný paket.

Duplicita paketu Příjemce obdržuje stále paket se stejným sekvenčním číslem. Odesílatel si myslí, že se paket k příjemci nikdy nedostal. Proč?

- Ztratí se potvrzení o přijetí paketu.

Vložení paketu Příjemce obdrží paket, který do spojení nepatří. Jak k tomu může dojít?

- Přijetí zpožděného paketu, který dorazil až po skončení jednoho a začátku dalšího separátního datového toku.
- Podvrhávání paketu útočníkem snažícím se narušit integritu sítě.
- Paket je chybou „zvláštně zmrzačen“ tak, že chybu nelze rozpoznat (např. je změněm bit v cílové IP adrese) a je směrován jinému příjemci.

Změna pořadí Vychází z designu počítačových sítí, kdy paketu mohou proudit různými cestami a s různým zpožděním. Data jsou *out-of-order* a příjemce je musí přeuspořádat.

13.4 Řízení toku

Řízení toku (*flow control*) je řízení komunikaci mezi dvěma uzly na straně koncového systému. Řízení zahlcení (*congestion control*) je řízení komunikace v rámci sítě. Alternativně, se za řízení toku považuje koncept detekce a korekce paketových chyb. Často jsou tyto termíny vykládány různě.

13.4.1 Detekce paketové chyby

Paketové chyby detekujeme pomocí sekvenčních čísel. Sekvenční číslo je jedinečný identifikátor paketu v rámci datového toku, který identifikuje jeho pořadí. Tímto poznáme ztrátu, duplicitu i změnu pořadí.

Jak rozpoznat ztrátu od zpoždění?

- **Timeout** – Může být fixní, ale v lepším případě se odvozuje od RTT.
- **Negativní potvrzování** – Paket jsem dostal (ACK) vs. paket jsem nedostal (NACK). Pokud je příčina zpoždění zahlcení, tak tento přístup linku ještě více zahltí.

13.4.2 Korekce paketových chyb

Pokud paket nedorazil, odesílatel ho pošle znovu. Tzv. *Automatic Repeat/Request* (ARQ) – Čeká se na ACK, když nepřijde do timeoutu, dojde k znovuzaslání. Princip tzv. klouzavého okna (*sliding window*). Existují 3 strategie.

Stop and wait

- Klouzavé okno o velikosti 1.
- Odesílatel pošle paket a čeká na potvrzení. Po přijetí potvrzení pošle další paket.
- Špatná efektivita využití pásma.

Go back n

- Buffer na straně odesílatele.
- Příjemce potvrzuje naposledy přijatým paketem (např. příjemce pošle ACK2 znamená, že dostal pakety 0, 1, 2).
- Plýtvání při znovuzaslání (např. odesílatel pošle 5 paketů, ztratí se první, musí poslat znovu všech 5).

Selective repeat

- Buffery jsou na obou stranách.
- Efektivní využití šířky pásma, ale složitější implementace.
- Příjemce potvrzuje:
  - Fast Retransmit – Příjemce posílá potvrzení s naposledy přijatým paketem.
  - Bitová maska – V ACK je zavedeno další políčko (bitová maska), které obsahuje informace o tom, co příjemce přijal a co ne.
  - NACK – Příjemce posílá ACK, pokud balík dostal a NACK, pokud nikoliv.

13.5 Řízení zahlcení

Zahlcení je detekováno, co se s tím dá dělat?

- Zvýšit kapacitu sítě,
- Snížit množství provozu.

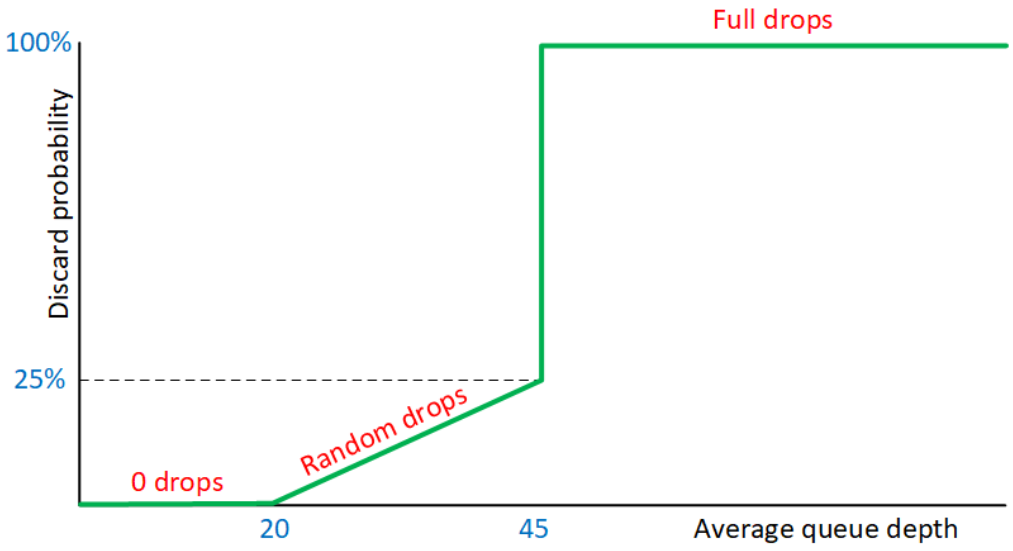
Jak se to dá dělat?

- Opravit zahlcený provoz (*repair*) – Je detekováno zahlcení a řešíme, co budeme dělat.
- Předcházet problému zahlcení (*avoidance*) – Snažíme se zahlcení předcházet (prevence).

13.5.1 Zahazování paketů (opravování provozu)

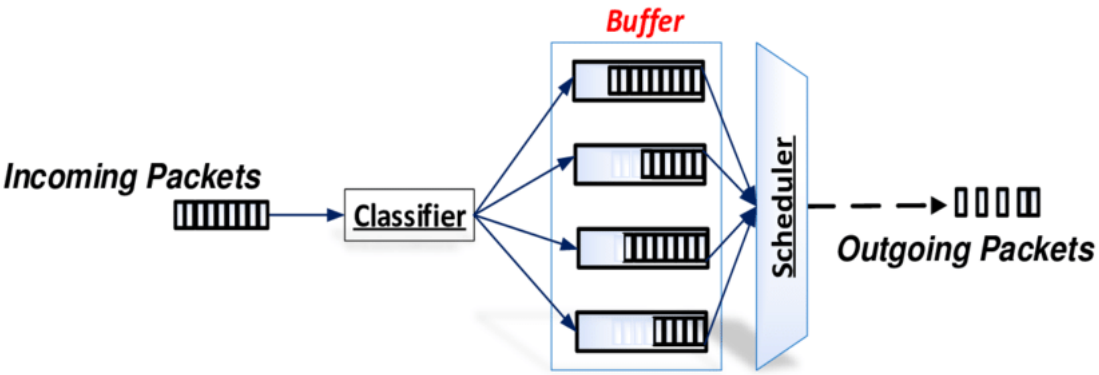
Každý uzel (najčastěji směrovač) v síti kontroluje délku své fronty, pokud příliš narůstá, tak začne preventivně některé pakety zahazovat (to však vede k jejich znovuzaslání).

RED (*Random Early Detection*) Algoritmus RED sleduje velikost fronty a zahazuje pakety na základě pravděpodobnosti. Pokud je fronta téměř prázdná, pravděpodobnost zahození je 0. Jak fronta roste, roste i pravděpodobnost zahození příchozího paketu. Když je fronta plná, pravděpodobnost dosáhne hodnoty 1 a všechny příchozí pakety jsou zahazovány.



Obrázek 13.2: Vizualizace RED.

WRED (*Weighted Random Early Detection*) Algoritmus WRED je rozšířením RED o přidání váh pro jednotlivé třídy paketů (máme frontu pro každou třídu). WRED nezahazuje všechny pakety se stejnou pravděpodobností, ale rozlišuje jejich důležitost (pomocí hodnot IP precedence nebo DSCP). Některé pakety chceme prioritizovat, např. VoIP (*Voice over Internet Protocol*), RTP (*Real-time Transport Protocol*).



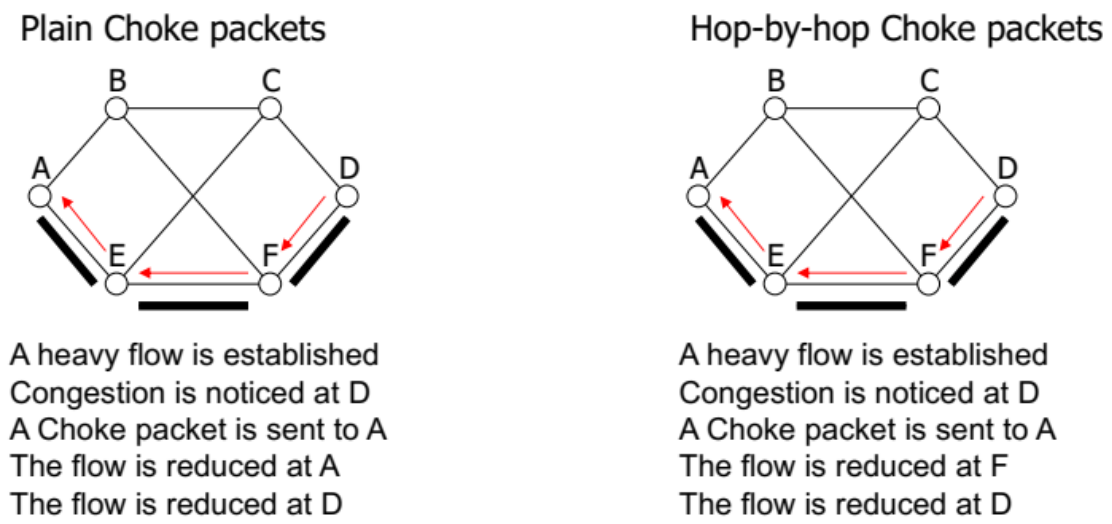
Obrázek 13.3: Vizualizace WRED.

13.5.2 Choke pakety (opravování provozu)

Příjemce v síti má problém zpracovávat provoz (nestíhá), tak dá vědět odesílateli (odešle choke paket), aby zpomalil.

Plain choke packets Viz obrázek 13.4. Uzel D detekuje zahlcení způsobené provozem od A. Uzel D pošle choke paket A. Proti zahlcení „zabojuje“ pouze odesílatel (A).

Hop-by-hop choke packets Viz obrázek 13.4. Uzel D detekuje zahlcení způsobené provozem od A. Uzel D pošle choke paket A. Každý uzel po cestě „bojuje“ proti zahlcení (F, E, A).



Obrázek 13.4: *Plain choke packets* vs. *hop-by-hop choke packets*.

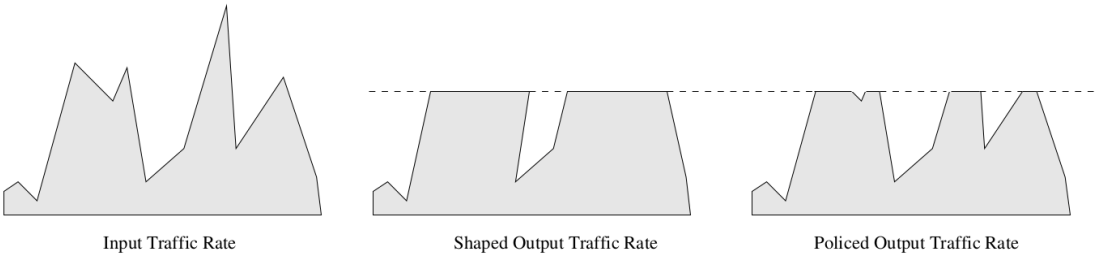
13.5.3 Ořezání a rozložení provozu (předcházení zahlcení)

Jedná se o předcházení zahlcení ze strany odesílatele. Odesílatel je „ohleduplný“ a myslí na to, aby příjemce nebyl zahlcen. Odesílatel má pevně stanovenou hranici kolik může maximálně posílat. Implementace pomocí tzv. *token bucket*.

Ořezání provozu (*policing*) Cokoliv nad hranici je „oříznuto“ a zahozeno.

Rozložení provozu (*shaping*) Cokoliv nad hranici je uloženo do bufferu a odesláno později, až je menší provoz.

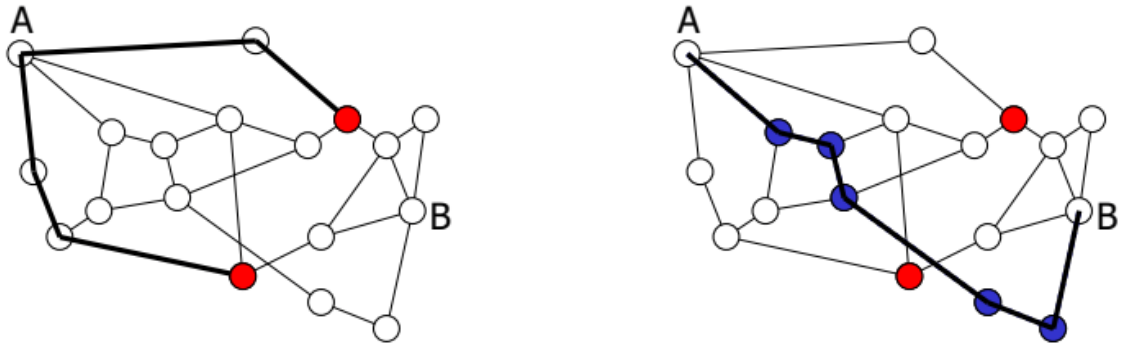
Čím se liší rozložení provozu (policing) a omezení provozu (shaping)?



Obrázek 13.5: Ořezání provozu vs rozložení provozu.

13.5.4 Rezervace (předcházení zahlcení)

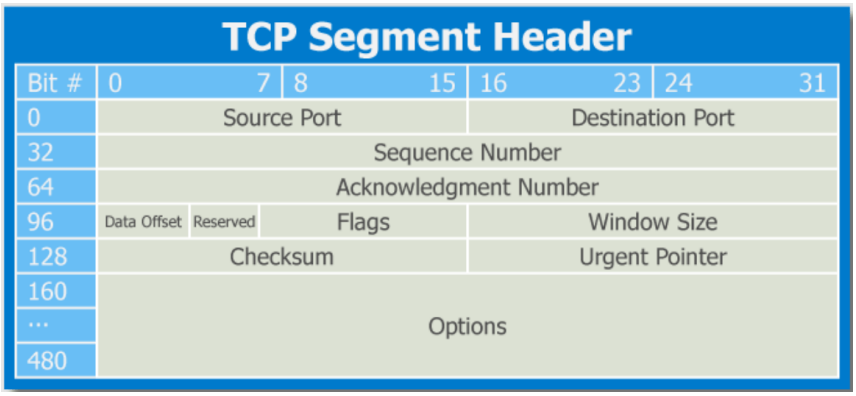
Zařízení v síti (směrovače) jsou schopni se domluvit a vytvořit si tzv. „virtuální okruh“ pro konkrétní komunikaci. Každé zařízení na okruhu ví o provozu a rezervuje potřebné pásmo (předem se deklarují kapacity). Je nutné více signaliace (režie).



Obrázek 13.6: Příklad rezervace.

13.6 TCP (*Transmission Control Protocol*)

- Garantuje spolehlivé doručení.
- Garantuje doručení v pořadí.
- Řízení toku a zahlcení.
- *Connection-oriented* – Navázání spojení pomocí *three-way handshake*.
- Pracuje s *byte stream* – nezohledňuje hranice aplikačních dat.
- Sekvenční číslo závisí na tom, kolik bajtů se posílá. Např. $seq = 92, data = 8\,B$, $seq = 100, data = 20\,B$, $seq = 120, data = 13\,B$, ...

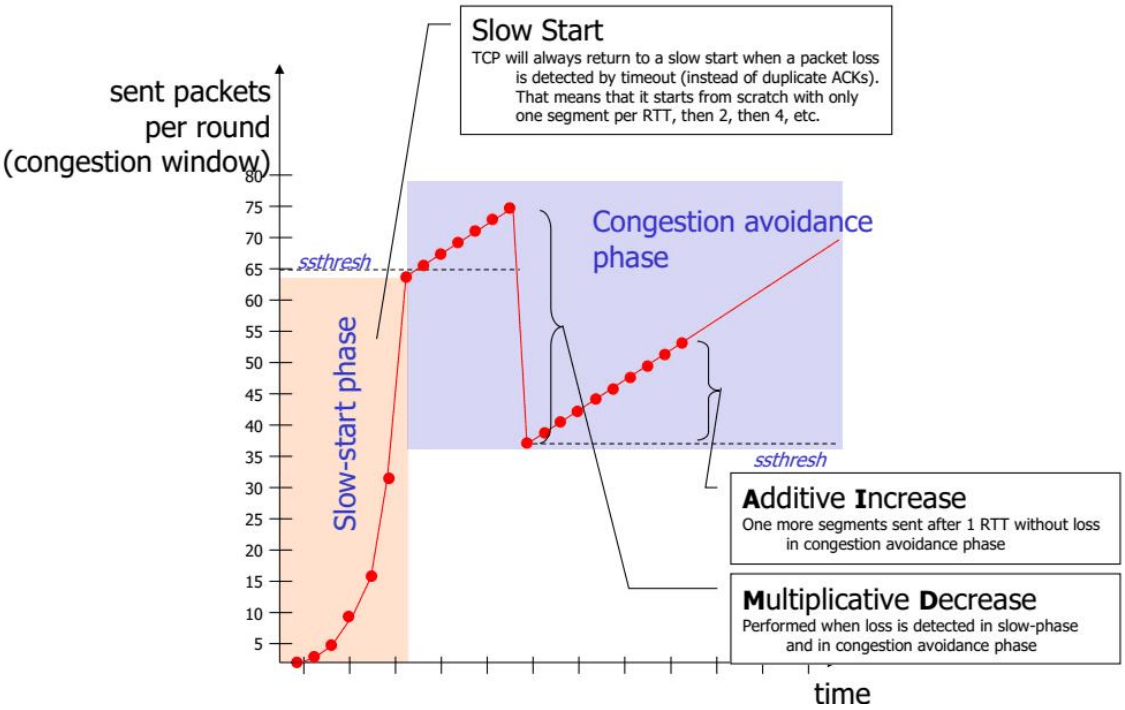


Obrázek 13.7: Hlavička TCP.

Řízení toku a zahlcení V TCP jsou rozlišovány 3 fáze přenosu dat.

- *Slow start* – Exponenciální zvyšování rychlosti odesílání až po dosažení nějaké hranice.
- *Congestion avoidance (additive increase)* – Lineární zvyšování počtu odeslaných paketů.
- *Congestion avoidance (multiplicative decrease)* – Skokové snížení rychlosti odesílání. Nastane, pokud je detekována ztráta paketu.

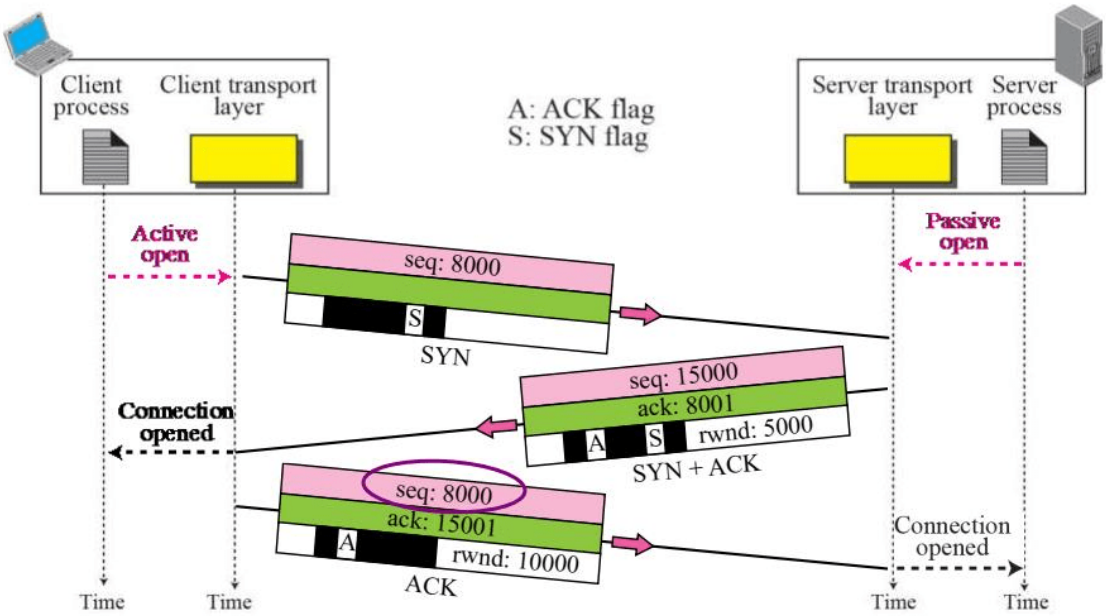
V TCP existuje spoustu algoritmů pro řízení toku a zahlcení, které implementují tento princip (Tahoe, Reno, New Reno, Vegas, CUBIC, Westwood, ...).



Obrázek 13.8: Jednotlivé fáze řízení zahlcení v TCP.

Nevýhody a problémy

- Nepodporuje *multi homing*.
- *Head of line blocking* – Pokud dojde ke ztrátě paketu, tak vše za ním je pozdrženo
- Velká režie potvrzování (až 35 % veškerého provozu je režie)



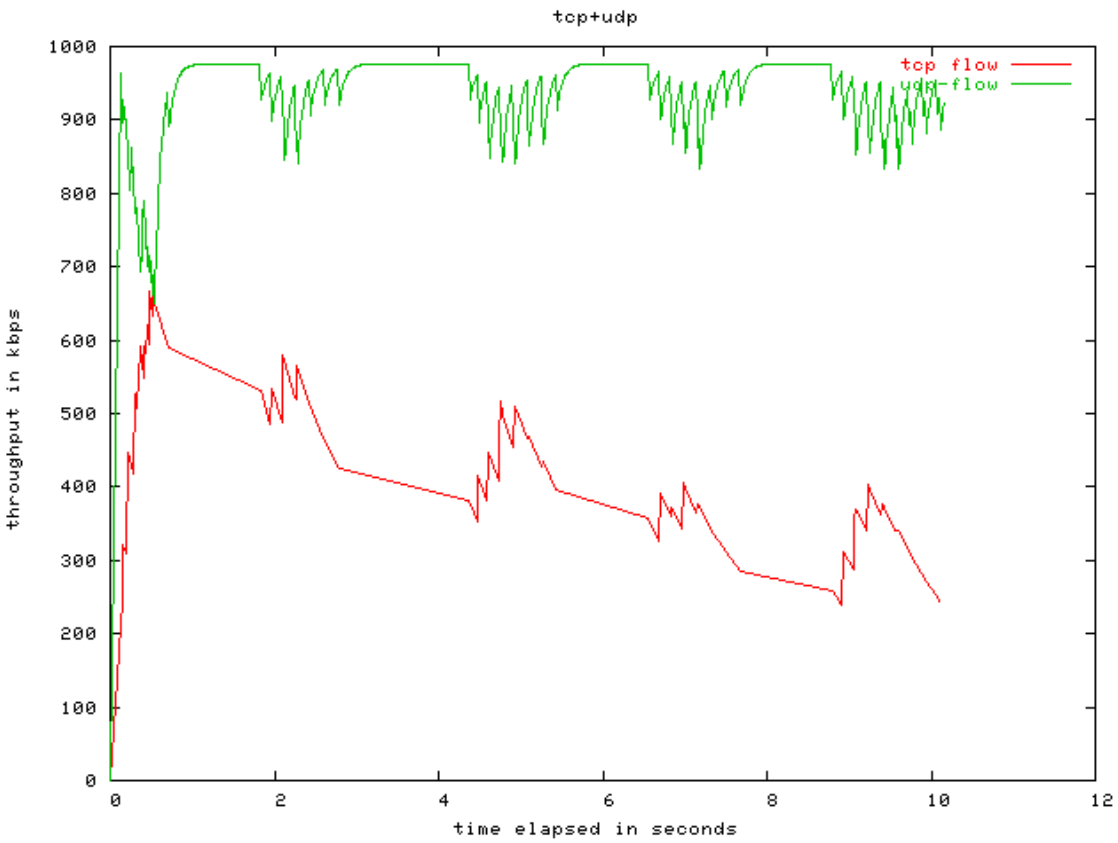
Obrázek 13.9: Zahájení spojení pomocí tzv. *Three-Way Handshake*.

13.7 UDP (*User Datagram Protocol*)

- Negarantuje spolehlivé doručení (*best effort*).
- Negarantuje doručení v pořadí.
- *Connection-less*.
- Pracuje s *byte stream* – nezohledňuje hranice aplikačních dat.
- Má minimální režii.

UDP Datagram Header				
Bit #	0 – 7	8 – 15	16 – 23	24 – 31
0	Source Port		Destination Port	
32	Length		Checksum	

Obrázek 13.10: Hlavička UDP.

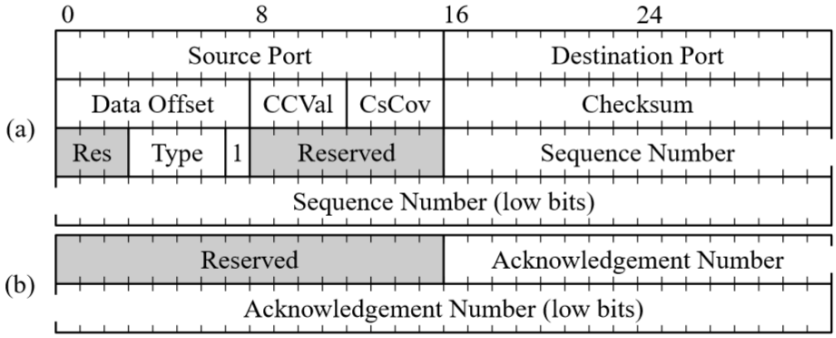


Obrázek 13.11: Problém UDP, kdy si pro sebe zabere celé pásmo, protože TCP se chová „zodpovědně“ a snižuje svůj provoz.

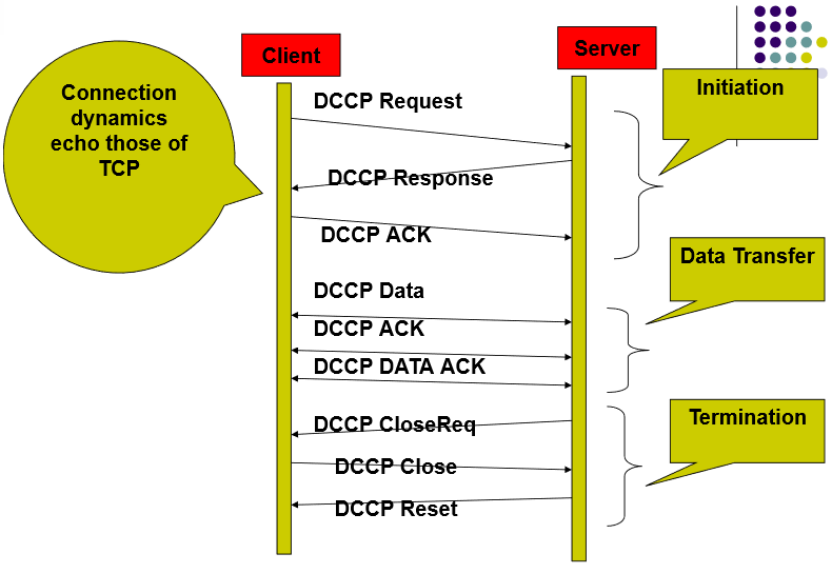
13.8 DCCP (Datagram Congestion Control Protocol)

- Cílem je zajistit podporu řízení zahlcení pro UDP.
- Negarantuje spolehlivé doručení (*best effort*).

- Negarantuje doručení v pořadí.
- *Connection-oriented* – Navázání spojení pomocí *three-way handshake*.
- Využití pro audio/video konference.



Obrázek 13.12: Hlavička DCCP. Sekvenční a potvrzovací čísla souvisí pouze s účtováním provozu kvůli řízení zahlcení, nikoliv pro zajištění spolehlivého přenosu.



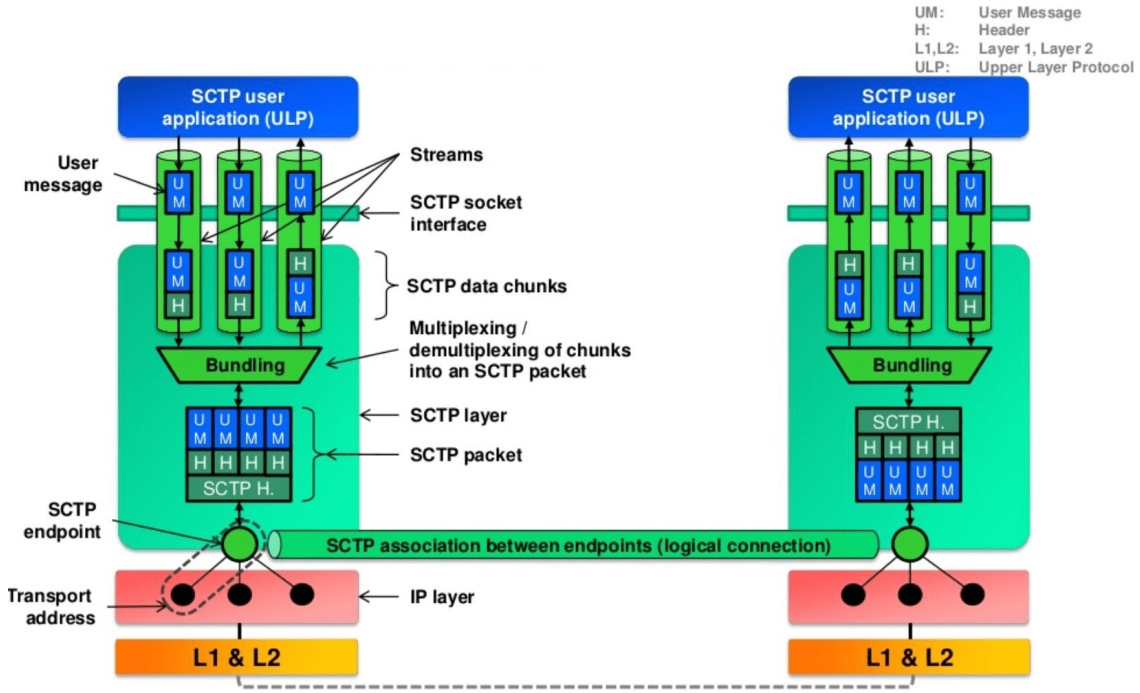
Obrázek 13.13: Komunikace pomocí DCCP. Obsahuje několik typů zpráv: Request, Response, ACK, Data, Data ACK, CloseReq, Close, Reset.

Jak je zahlcení realizováno V potvrzeních se nastavují tzv. ECN (*Explicit Congestion Notification*) bity. Pomocí nich dává příjemce odesílateli vědět, jak moc je zahlcen. V průběhu navazování spojení se pomocí CCID (*Congestion Control ID*) dohodne, jaký typ řízení zahlcení se bude používat (algoritmy jako pro TCP, nebo vlastní).

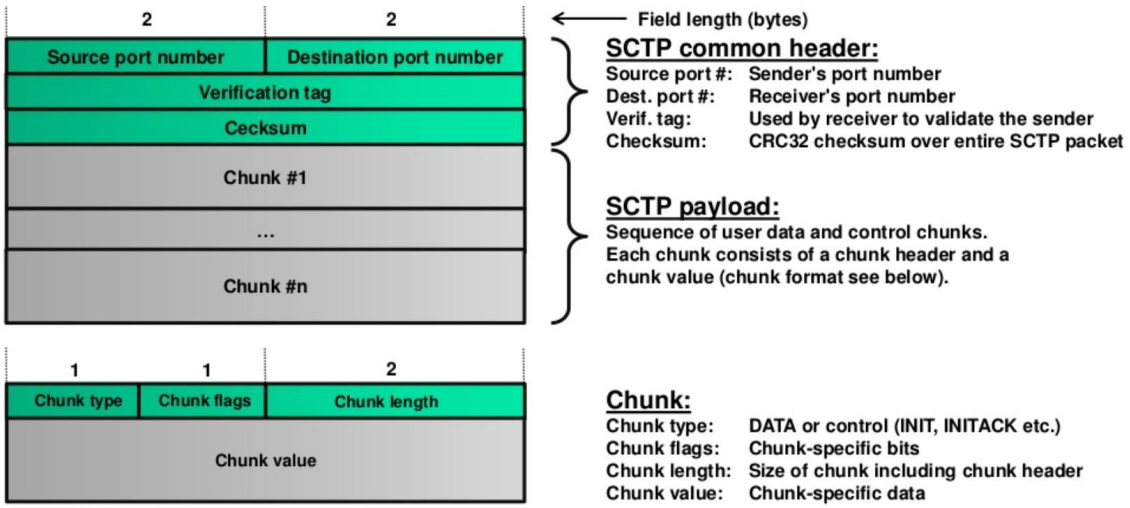
13.9 SCTP (*Stream Control Transmission Protocol*)

- Garantuje spolehlivé doručení.
- Garantuje doručení v pořadí.

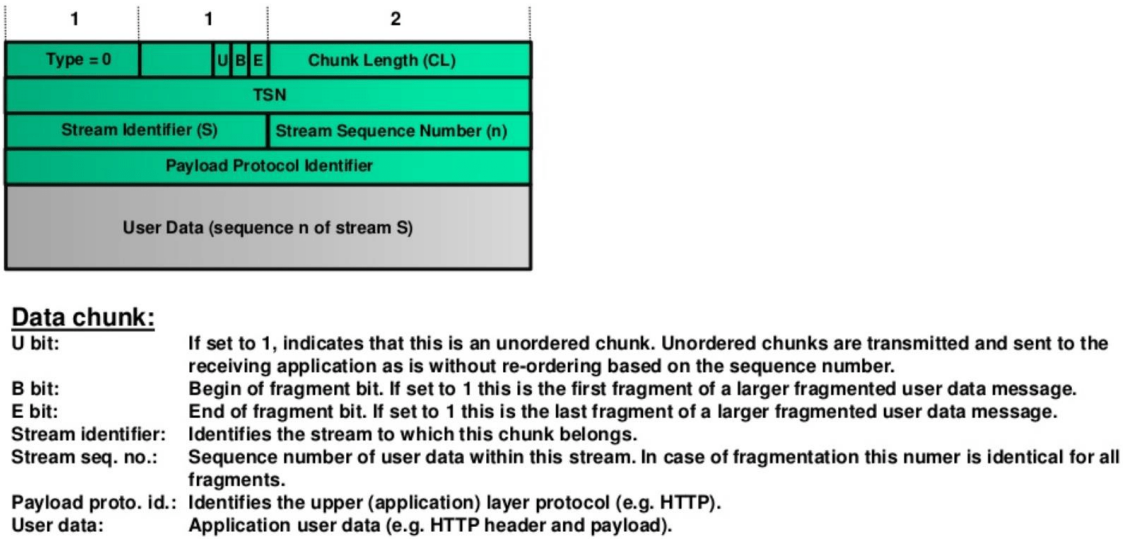
- Pracuje s *message stream* – Respektuje hranice aplikačních dat.
- *Connection-oriented* – Navázání spojení pomocí *four-way handshake*.
- *Path MTU discovery* – Obrana proti fragmentaci (od zdroje k cíli jsou prozkoumány MTU všech uzlů).
- Netrpí problémem *head of line blocking*.
- Podporuje *multi homing*.
- Řízení zahlceni.
- Aplikace je třeba přeprogramovat, aby místo TCP socketů využívali SCTP sockety.



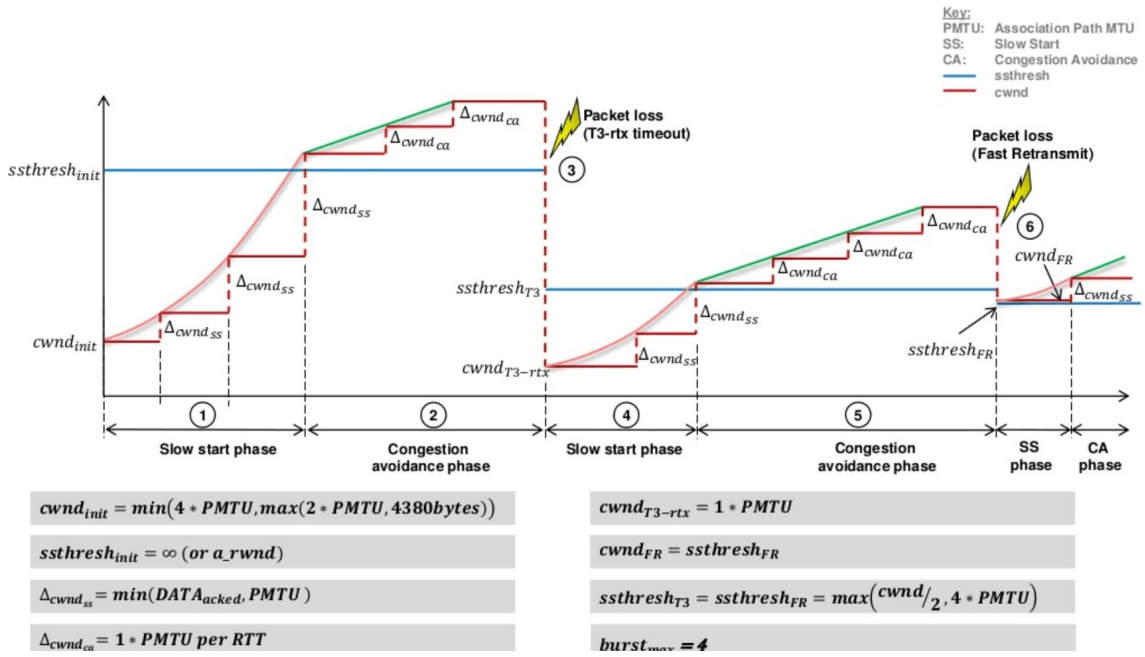
Obrázek 13.14: Komunikace pomocí SCTP. Komunikace s aplikací probíhá přes sockety. Každé aplikační zprávě je přidán základní hlavičkou – vzniká *data chunk*. Několik data chunku je zabaleno dohromady a opatřeno SCTP hlavičkou – vzniká SCTP paket. Ten je poté možno posílat přes více IP spojení.



Obrázek 13.15: Hlavička SCTP. SCTP paket a *data chunk*.



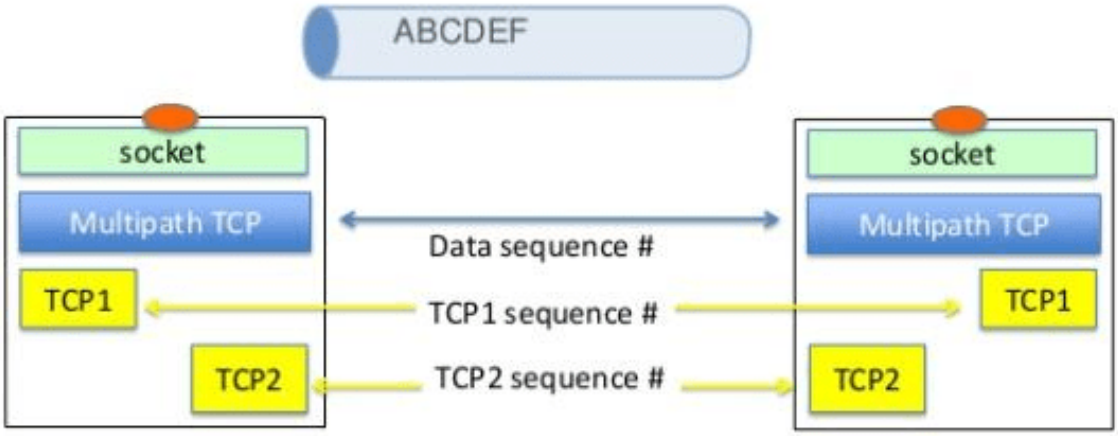
Obrázek 13.16: Detailní hlavička *data chunk*. Identifikátor streamu a identifikátor paketu v rámci streamu.



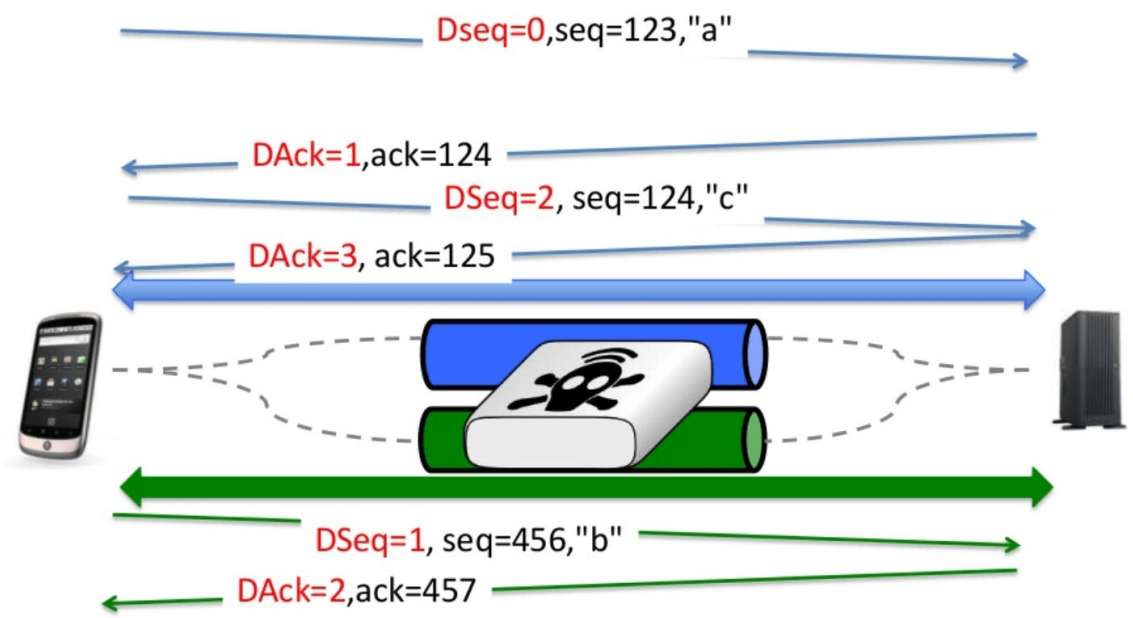
Obrázek 13.17: Řízení zahlcení (*slow start*, *congestion avoidance (additive increase)*, *congestion avoidance (multiplicative decrease)*) je podobné jako TCP Westwood.

13.10 MP-TCP (*Multipath TCP*)

- Cílem je vzít co nejvíce vlastností z SCTP a poskytnout je v TCP bez toho, aniž by se musely přeprogramovávat aplikace (má obdobné sockety jako TCP).
- Má menší signalizační náročnost oproti SCTP.
- Chová se jako TCP pouze, podporuje *multi homing* a má vlastní řízení zahlcení.
- Rozšíření TCP hlavičky o tzv. *options*.



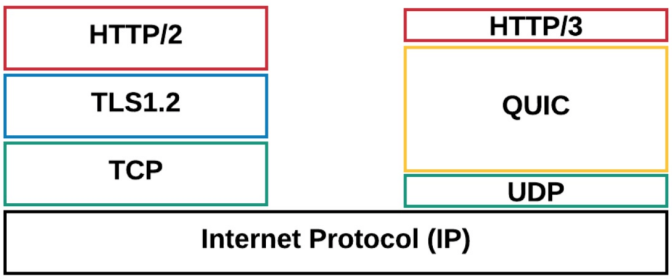
Obrázek 13.18: MPTCP komunikace, rozdělení aplikačních dat na paralelní TCP streamy. Rozlišujeme datová sekvenční čísla a sekvenční čísla TCP.



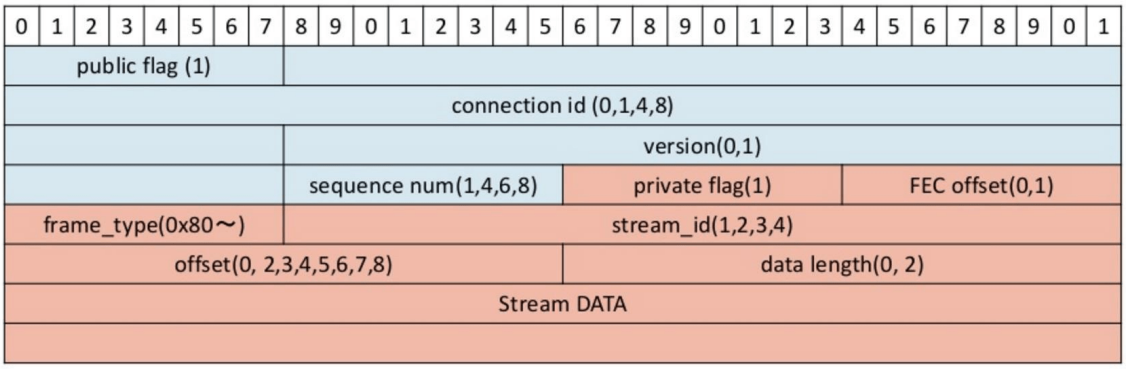
Obrázek 13.19: MPTCP řízení toku. Dvě TCP komunikace a posílání řetězce „abc“. Navázání spojení podobné jako v TCP. Může vypadnout i celý komunikační kanál.

13.11 QUIC (*Quick UDP Internet Connections*)

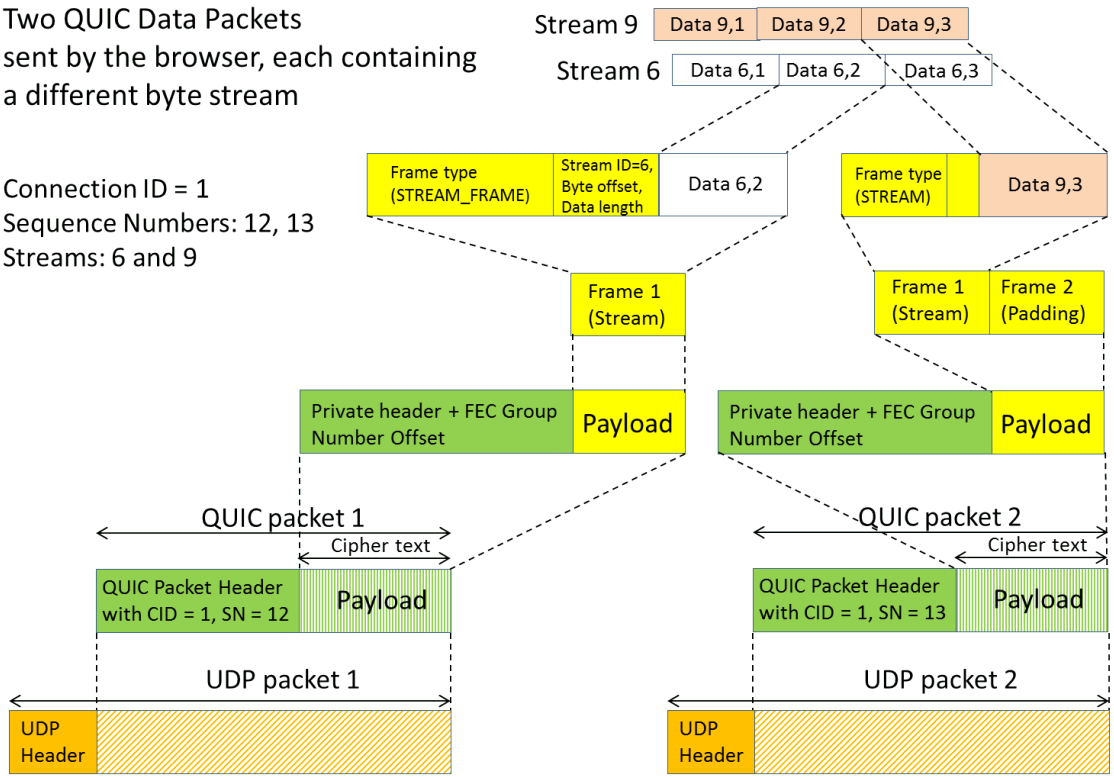
- Motivace pro QUIC: dnes většina provozu probíhá šifrovaně přes HTTPS. Navázání bezpečného HTTPS spojení má velkou režii (HTTPS = TCP + SSL + HTTP). Až $3 \times$ RTT přes započetím komunikace!
  - TCP: syn, syn ack, ack
  - TLS: client hello, server hello, change cipher spec, encrypted handshake message
- Staví nad UDP (aby všechny zařízení v internetu nemuseli implementovat nový transportní protokol).
- Paralelní datové streamy, aby bylo zabréněno *head of line blocking*.
- Má zabudovanou *Forward Error Correction* (FEC), přidává bitovou redundanci, tím umožňuje velké množství ztracených paketů dopočítat. Pokud to není možný, tak „chytrý“ selective repeat.



Obrázek 13.20: Standardní TCP stack vs. QUIC.



Obrázek 13.21: Hlavička QUIC. Spousta políček má proměnlivou velikost.



Obrázek 13.22: Složení QUIC zprávy.