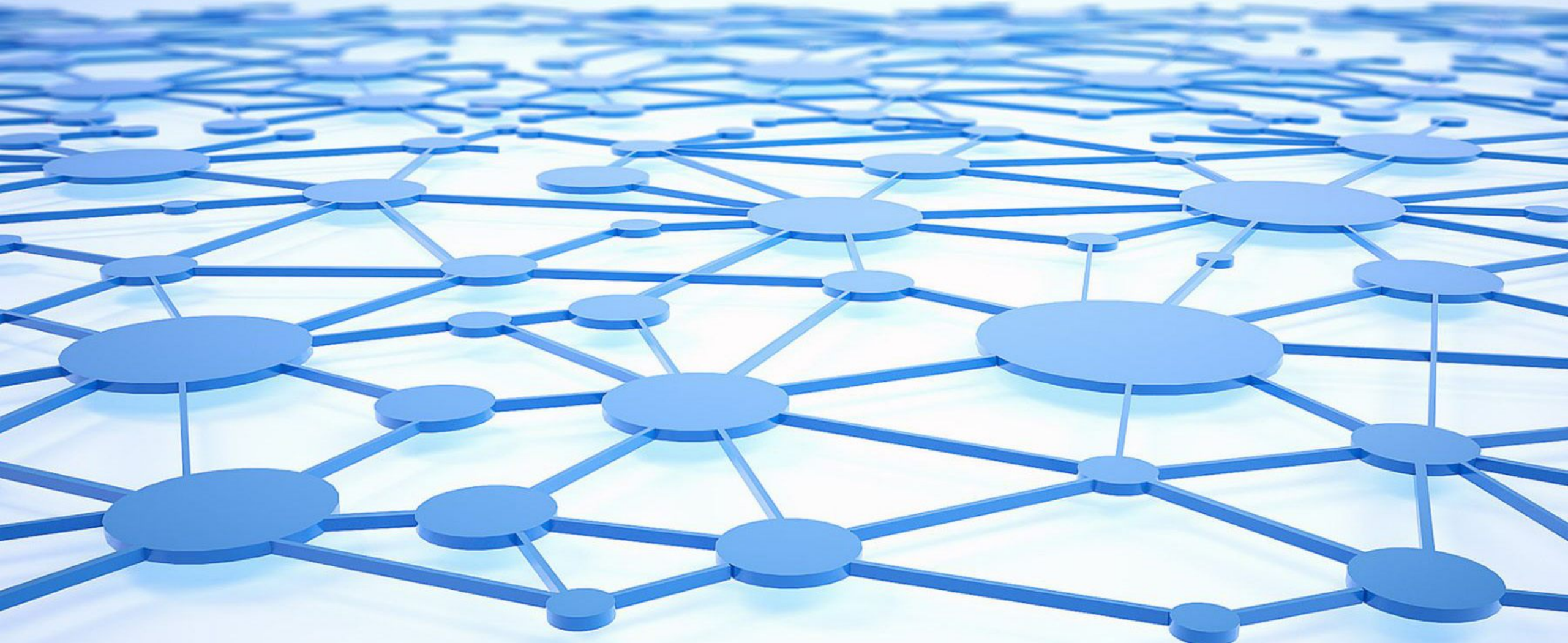
Úvod do počítačových sítí

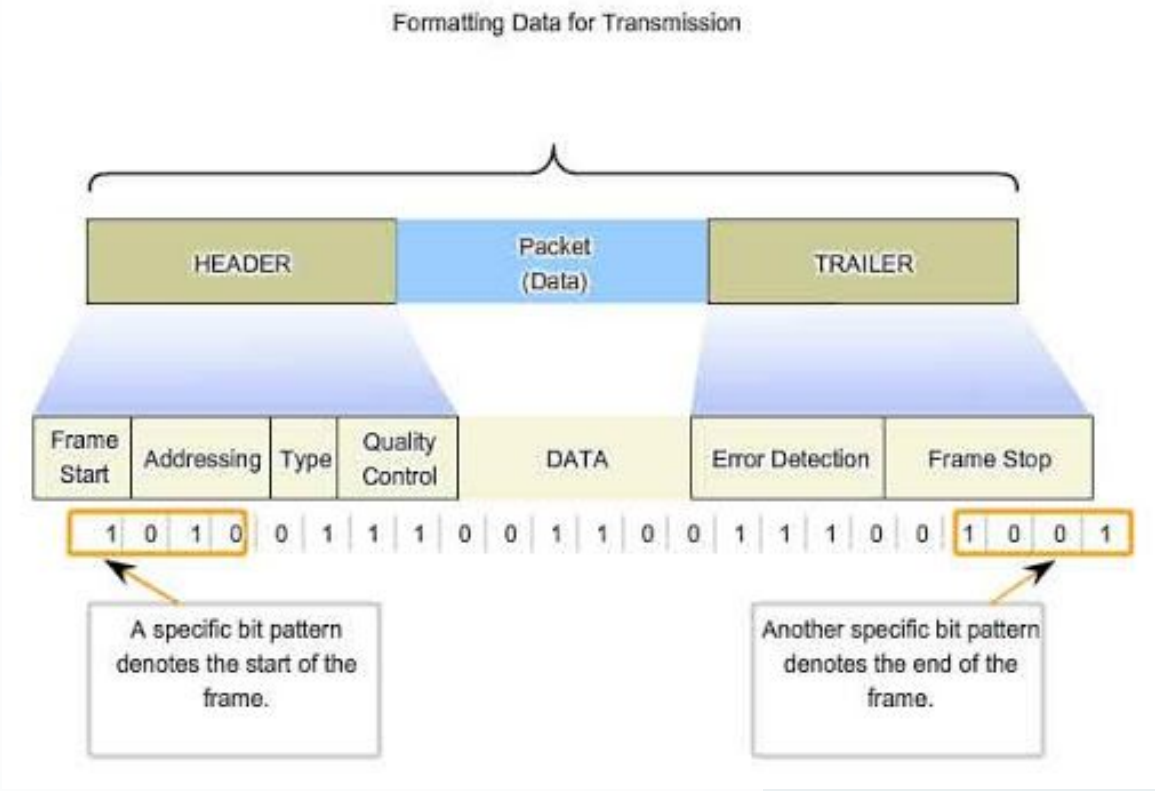
Přednáška 5 (2021/2022)

ver. 2021-10-26-01



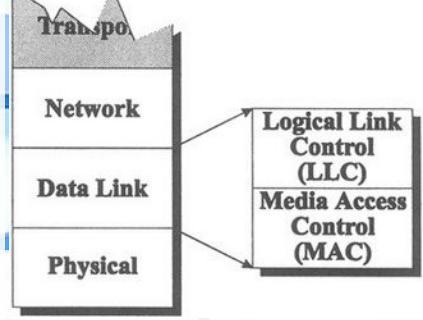
Linková vrstva: L2

- Základní funkce L2 – linkové vrstvy
  - Přijímá data od L1 vrstvy / předává data L1 vrstvě (bity)
    - Využívá služeb L1 k přenosu rámců
  - Provádí rozdělení na rámce
  - Provádí kontrolu správnosti přenosu
  - Detekuje a řeší chyby v přenosu
  - Řídí přístup k komunikačnímu kanálu
  - Řídí rychlost přenosu
  - Zajišťuje přenos rámců v rámci LAN / mezi sousedními uzly
  - Předává data L3 vrstvě / přijímá data od L3 vrstvy (pakety)
    - Poskytuje služby L3 – přenos data, řešení chyb, řízení rychlosti



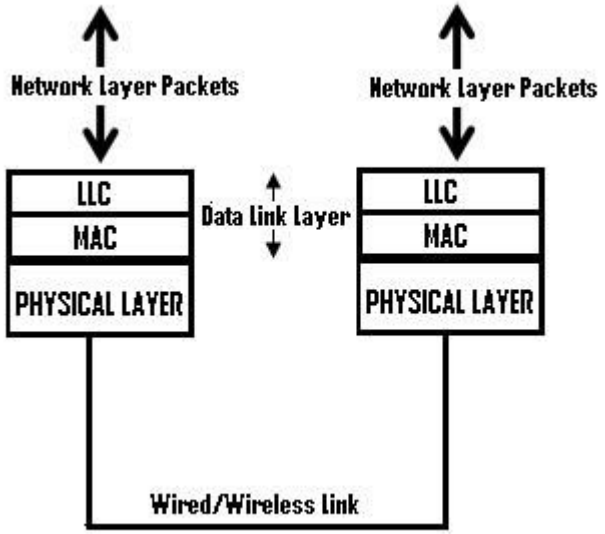
zdroj: http://www.highteck.net/EN/DataLink/Data_Link_Layer.html

L2: Dělení linkové vrstvy



zdroj: https://networkencyclopedia.com/logical-link-control-llc-layer/

- L2 v ISO/OSI je „přetížená“
  - Má více rozdílných funkcí
- Začala se pro přehlednost dělit na dvě části – dle funkce
  - MAC - Media Access Control
    - fyzické adresování,
    - řízení přístupu k médiu
    - Fragmentace/defragmentace data (bit → rámec)
  - LLC - Logical Link Control
    - řízení toku
    - zabezpečení proti chybám
    - Zabalování / rozbalování L3 paketů



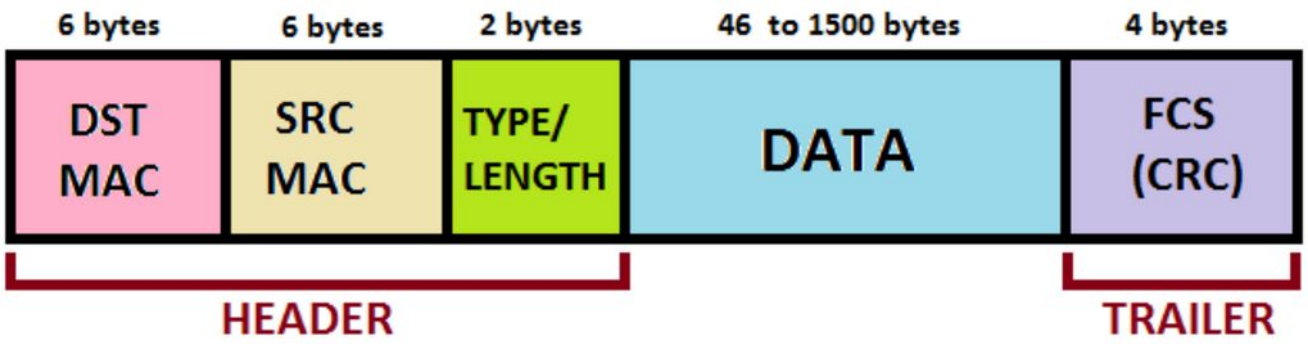
zdroj: http://computernetworkingsimplified.in/data-link-layer/components-data-link-layer-llc-mac/

L2: Rámce a jejich význam

- Rámec je základní datová jednotka L2 vrstvy
- Rámce mohou být
  - Datové – obsahují data, které chceme přenést
  - Řídící – obsahují příkazy, potvrzení či další informace nutné k řízení přenosu
- Framing – rozdělení streamu 0 a 1 od L1 na rámce – frame
- L2 už „nepřenáší“ jednotlivé bity, ale celé rámce
  - Která se z 0 a 1 skládají
- Konkrétní rámec je vázaný na konkrétní L2 protokol – je vždy jiný
  - Ethernet, Token Ring, PPP, SLIP, ….

L2: Rámce a jeho struktura

- Rámec se liší dle konkrétního L2 protokolu, ale základní struktura je stejná
- Hlavička
  - Cílová L2 adresa
  - Zdrojová L2 adresa
  - Identifikace protokolu / délka rámce /… liší se dle protokolu
- Data
  - Datový rámec - L3 paket
  - Řídící rámec – L2 data pro řízení přenosu, např ACK potvrzení správnosti přenosu
- Patička – FCS - frame check sequence
  - Zabezpečení rámce – ověření správnosti přenosu
    - Parita, checksuma, CRC, …



zdroj: https://www.bitforestinfo.com/2018/01/code-ethernet-ii-raw-packet-in-python.html

L2: Detekce rámce

- Detekce rámce je jednou ze základních funkcí L2 vrstvy
  - Od L1 dostává nestrukturovaný stream/proud 0 a 1
- Metody detekce záleží na typu přenosu
  - Odpočítat rámec
    - Pokud data jsou vždy těsně za sebou (bez mezer) a mají pevně danou strukturu
    - Například v rámci T nebo E kanálů
    - Podobný princip jako časový multiplex
  - Vyznačení začátku a konce rámce
    - Pomocí speciální sekvence bitů/znaků/bytů označíme začátek konec
      - Ale musí řešit transparentnost přenosu – tedy jak tyto speciální značky přenést v datech ….
  - Vyznačení začátku a dopočítání konce rámce
    - Stejně jako v předchozím případě vyznačím začátek rámce a jelikož znám délku rámce, tak už jen odpočtu příslušný počet bitů
    - Výhoda je, že šetřím kapacitu kanálu, protože nepotřebuji ukončovací značku rámce
    - Délka může být pevně daná nebo přenášená v datech
    - Na rozdíl od „Odpočítávání rámec“ data nemusí jít bezprostředně za sebou

L2: Typy přenosů

- Podle toho jak přenášená data (0 a 1) nahlížíme, můžeme na přelomu L1 a L2 definovat tři typy přenosu
  - Bitové orientovaný
    - Neskládám z bitů jednotlivé znaky, ale pracuji přímo s jednotlivými bity
  - Znakově orientovaný
    - Jako základní jednotku neberu 1 bit, ale pracuji vždy s N-ticí bitů => znakem
    - Znak může být různě dlouhý - dle protokolu (typicky 7-8 bit, ale klidně např 5)
  - Bytově orientovaný
    - Speciální případ znakově orientovaného přenosu, kdy délka jednoho znaku je 8 bitů – 1 byte

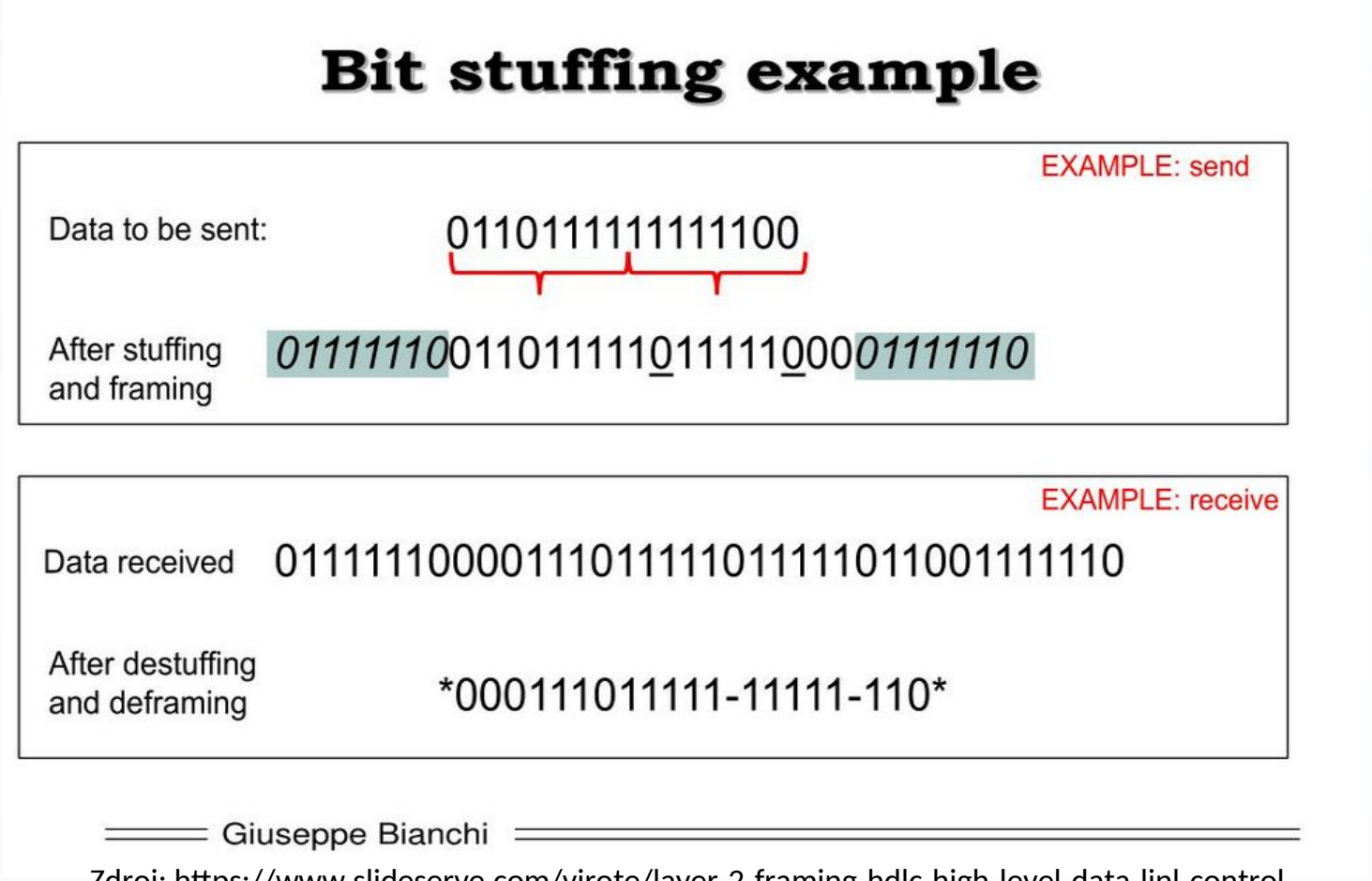
L2: Bitový přenos

- Bitový orientovaný přenos bere jako základní jednotku 1 bit
- Tím že přenášíme jednotlivé bity, musí hranici tvořit nějaká bitová sekvence
  - Což vypadá stejně jako u znakově orientovaného protokolu, ALE
  - „křídlová značka“ – začátek rámce je sekvence bitů, které postupně dostáváme jeden po druhém
    - Často označovaná jako „flag“
  - Další data stále přenášíme po jednotlivých bitech bez ohledu na délku křídlové značky
- Příklad křídlové značky pro protokol HLDC
  - 01111110
  - Tedy nula – šest jedniček – nula => začátek rámce

L2: Bitový přenos – transparentnost přenosu

- Stejně jako u znakově orientovaného přenosu vzniká problém transparentnosti přenosu
- Máme křídlovou značku/flag
  - 01111110
- Jak jej přenést v datech ?
  - Uvodit jej nějakým znakem nemůžeme – máme jen jednotlivé bity
  - Uvodit jej jediným bitem také ne, protože to může být jen 0 a 1
- Řešením je „vkládání bitů“
  - Pokud mám křídlovou značku 01111110 musím zajistit aby se mi nikde v datech nevyskytla stejná sekvence
  - Tedy aby tam nebylo 111111 – šest jedniček za sebou
  - Řešením je za každou „pátou jedničku“ uměle vložit nulu
    - POZOR – jen v datech
    - Nula je vložena automaticky na straně odesilatele (pokud se v datech vyskytne pět jedniček)
    - Nula po sekvenci pěti jedniček je na straně přímce odebrána (pořad se bavíme jen v datech)

L2: Bitový přenos – transparentnost přenosu -příklad

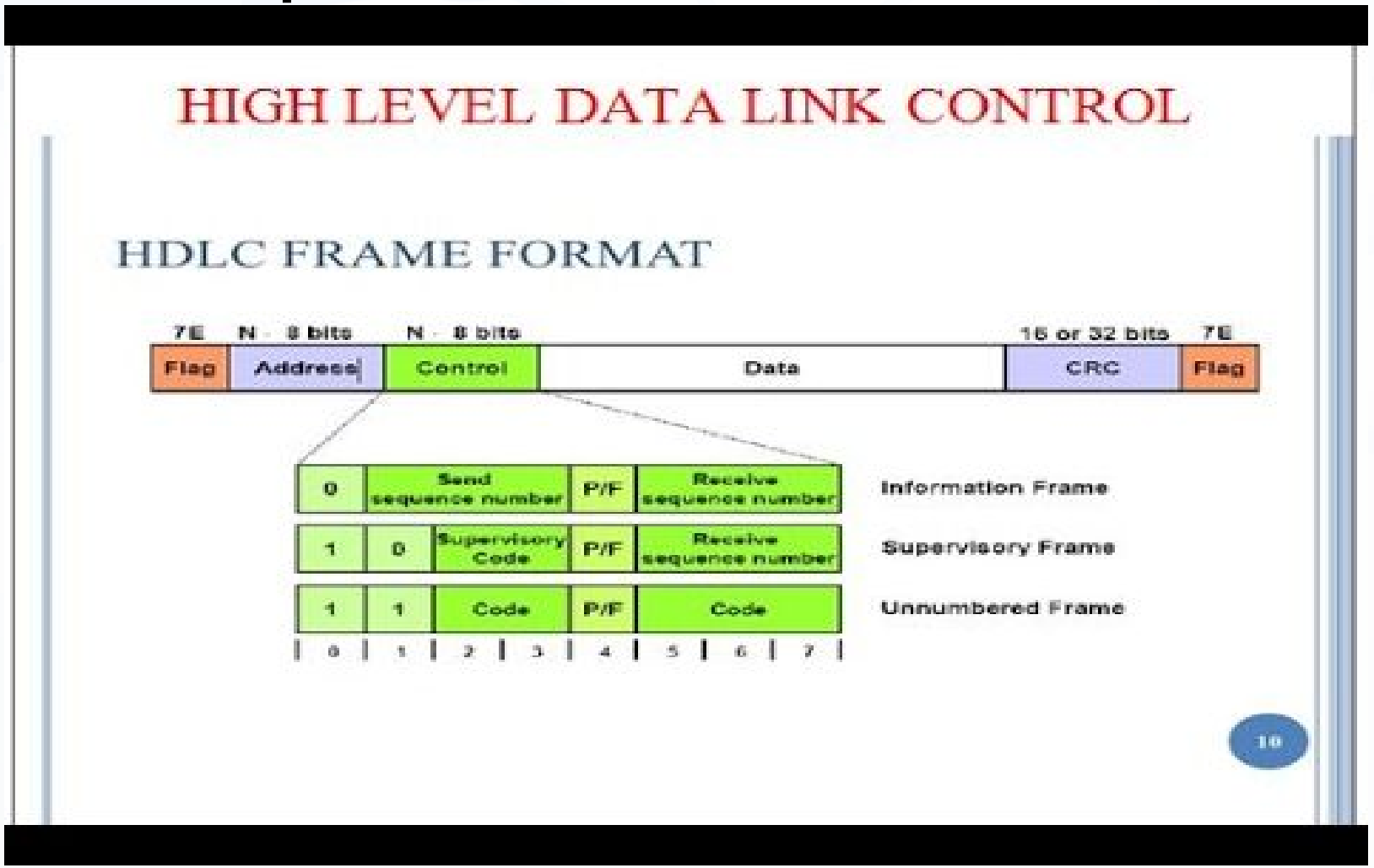


Zdroj: https://www.slideserve.com/virote/layer-2-framing-hdlc-high-level-data-linl-control

L2: Bitově orientovaný přenos příklad - HLDC

- HLDC - High-Level Data Link Control
- Bitově orientovaný protokol pro Point-to-Point i Point-to-MultiPoint spoje
  - Sice už se dnes nepoužívá, ale sloužil jako „podklad“ nebo inspirace pro mnoho dalších protokolů
- Protokol používá křídlovou značku **01111110**
  - Křídlová značka na konci rámce může být i začátkem dalšího rámce => šetříme data
- Používá tři „typy“ rámců“, které se rozlišují v rámci části rámce „control“
  - Informační - 0 (uživatelská data)
  - Řídící - 10 (řídící informace)
  - Nečíslované – 11 (vše ostatní)
- Obsahuje zabezpečení – FCS – patička
  - Používá **16 nebo 32 CRC** (budeme mít v dále v přednáškách)

L2: Bitově orientovaný přenos příklad – HLDC - rámce



Zdroj: https://www.youtube.com/watch?v=5BqMcgxIsws

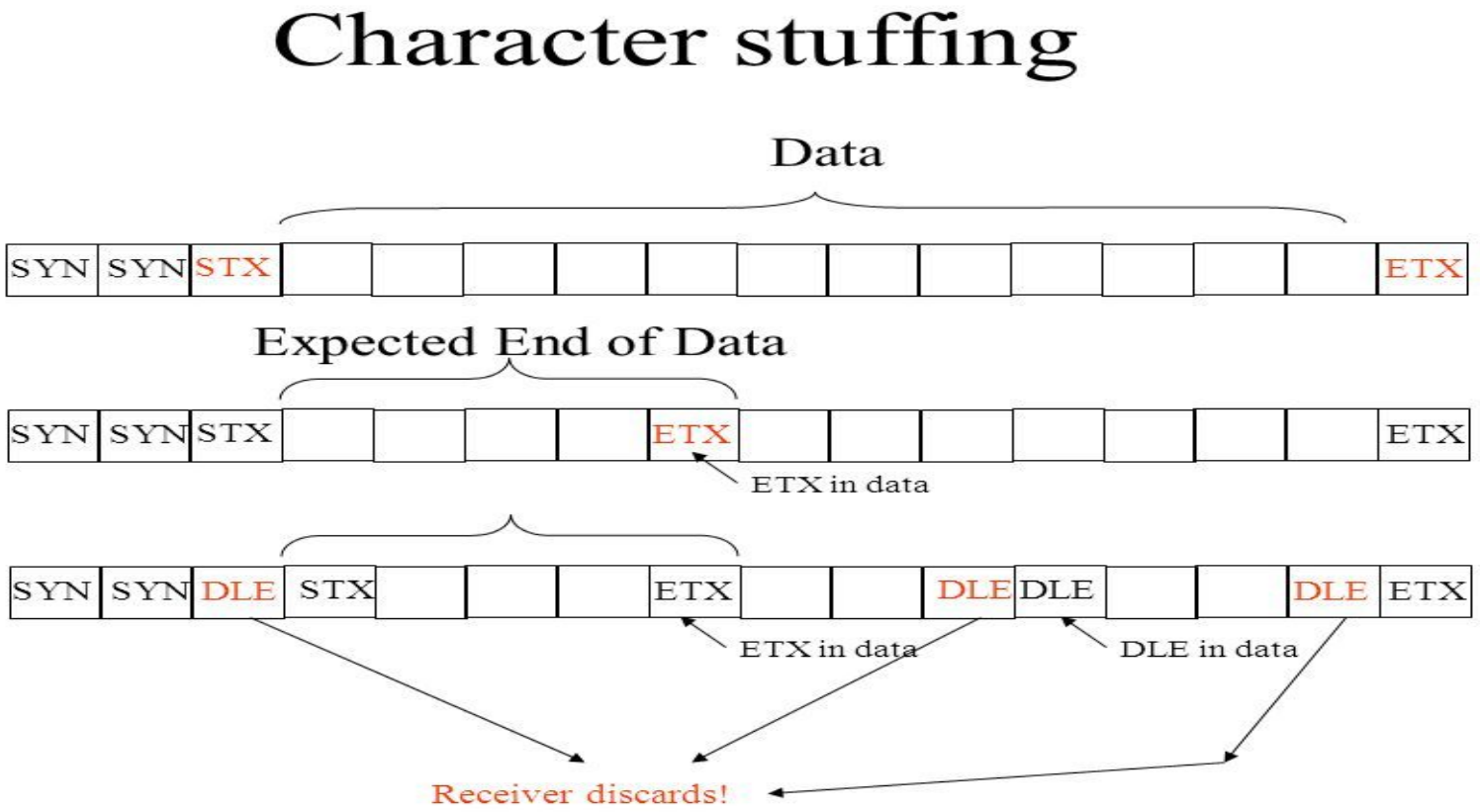
L2: Znakově orientovaný přenos

- Základní jednotka kterou detekuji je 1 znak
- 1 znaků je tvořen N-bity, kde N může být různé dle konkrétního protokolu
  - Např 8, 7, 5 …
- Začátek a konec rámce je uvozen speciálními znaky
  - Takzvané řídící znaky
- V případě detekce rámce dle označení začátku a konce potřebujeme dva řídící znaky
  - Začátek rámce - např STX nebo BOF
  - Konec rámce – např ETX EOF
- Rámec pak vypadá při přenosu
  - **STX**|HLAVIČKA|DATA|FCS|**ETX**

L2: Znakově orientovaný přenos - transparentnost přenosu

- Řídící znaku (např STX A ETX) nám umožní detekovat začátek a konec rámce
  - STX|HLAVIČKA|DATA|FCS|ETX
- ALE co když se vyskytnou v datech ?
  - STX|HLAVIČKA|DATA**STX**DATA|FCS|ETX
  - Nastává problém, protože nevím zda je to začátek nové rámce nebo jen data!
- Je nutné zajistit „transparentnost“ přenosu – tedy jak bezpečně přenést i řídící znaky
- U znakově orientovaných přenosu používáme „escapování“
  - Tedy další řídící znak – např DLE – který říká, že to co následuje jsou JEN data a ne příkaz
  - STX|HLAVIČKA|DATA**DLESTX**DATA|FCS|ETX
- Po vyřešení STX a ETX nám ale vznikl nový problém – jak přenést DLE ?
  - STX|HLAVIČKA|DATA**DLESTX**DATA**DLE**DATA|FCS|ETX
- Vyřešíme stejně, tedy opět escapováním
  - STX|HLAVIČKA|DATA**DLESTX**DATA**DLEDLE**DATA|FCS|ETX
- DLE se vkládá při odeslání do dat a při přijmu se z nich zas na vstupu odebírá

L2: Znakově orientovaný přenos - transparentnost přenosu - příklad



Zdroj: https://slideplayer.com/slide/5221152/

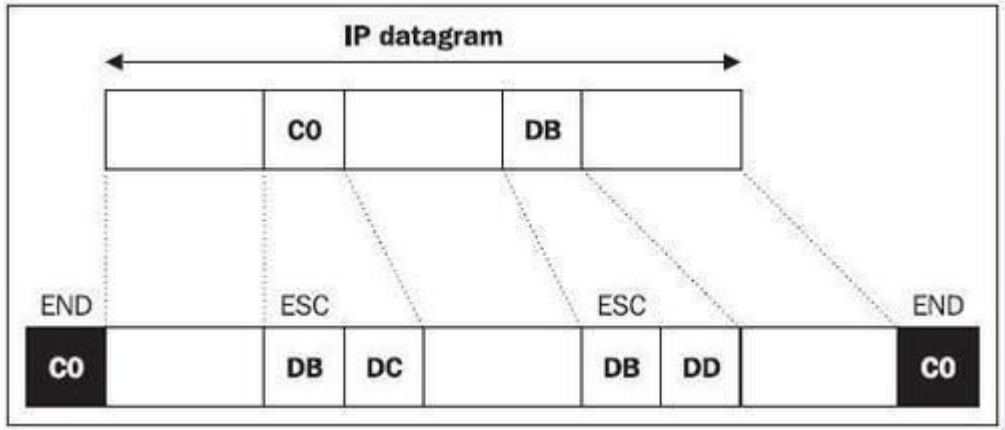
L2: Znakově orientovaný přenos příklad SLIP

- Příkladem znakově orientovaného protokolu je např SLIP
  - Serial Line Internet Protocol
  - Protokol pro zapouzdření IP a pro jeho následný přenos po sériové lince
  - Použité značky
    - Délka 8bit
    - Bez parity

Hex value	Dec Value	Oct Value	Abbreviation	Description
0xC0	192	300	END	Frame End
0xDB	219	333	ESC	Frame Escape
0xDC	220	334	ESC_END	Transposed Frame End
0xDD	221	335	ESC_ESC	Transposed Frame Escape

zdroj: https://en.wikipedia.org/wiki/Serial_Line_Internet_Protocol

  - Příklad rámce
    - Rámec „začneme“ značkou END
    - Pokud v datech máme ESC, pošleme ESC, ESC_ESC
    - Pokud v datech máme END, pošleme ESC, ESC_END
    - Pokud mám v datech ESC_END nebo ESC_ESC nic se neděje, protože k jejich aktivaci je třeba znak ESC před



zdroj: https://subscription.packtpub.com/book/networking_and_servers

L2: Bytově orientovaný přenos

- Bytově orientovaný protokol pracuje s pevně danou sekvencí bitů – bytem – jako znakově orientovaný protokol
  - Tedy se dá říct, že „znak“ má fixní délku 8 bitů
- Ale data neinterpretuje jako znaky, ale pracuje se sekvencí bitů
- Fakticky vždy máme jednotlivé bity – jde o interpretaci
  - Samostatné bity
  - Různě dlouhé sekvence bitů tvořící znak
  - Pevně daná sekvence bitů o velikosti 8 bitů – jeden byte
- Používá křídlovou značku – tedy sekvenci bytů – např **8x 10101010** pro Ethernet
- Stejná křídlová značka může být použita pro začátek i konec přenosu
- Je méně efektivní z hlediska využití přenosové kanálu než bitový přenos
  - Logicky, i pokud chci přenést menší data než 1 byte, přenese se 1 byte

L2: Bytový přenos - transparentnost přenosu

- Stejně jako i bitového a znakového přenosu musí být i u bytového přenosu zajištěna transparentnost přenosu – tedy možnost přenosu křídlového bytu/flagu v datech
- Transparentnost se zajišťuje podobně jako v znakově orientovaného protokolu a tedy „escape“ bytem – v tomto případě je to celý byte
- Identicky i přenos „escape“ značky je, stejně jako u znakově orientovaného přenosu, řešen zdvojením „escape“ bytu
- Příklad
  - Flag 01111110, escape 11111111, data 01111110|011001100|11111111
  - Přenos bude
  - 01111110|**11111111**|01111110|011001100|**11111111**|11111111|01111110

L2: Bytově orientovaný přenos příklad Ethernet

- Příkladem L2 bytově orientovaného protokolu je Ethernet
  - Jeden z nejpoužívanějších protokolů dneška
- Existuje dnes více variant
  - **Ethernet II / IEEE802.3 Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications** / 802.2 SNAP / 802.2 LLC
  - Základní rámec je velice podobný, liší se v drobnostech
    - Preambule
      - 10101010 – slouží pro synchronizaci
      - SOF – Start Of Frame 10101011 – křídlová značka
    - Cílová adresa: MAC
    - Zdrojová adresa: MAC
    - Délka / Typ
      - **IEEE 802.3 – délka dat (max 1500)**
      - **Ethernet II – type rámce (> 1500)**
    - 802.2 Header + Data / Data
      - IEEE 802.3 - 802.2 Header + Data
      - Ethernet II - Data
    - FCS: zabezpečení

Zdroj: http://wiki.ciscolinux.co.uk/index.php?title=Internet_Standard_organizations

L2: Chyb v přenosu

- V reálném přenosu mohou vznikat chyby téměř vždy
- Zdroje chyb
  - Bílý – tepelný šum
    - Pohyb elektronu / útlum signálu – ten prostě je a nic s tím neuděláme
    - Je trvale přítomen
  - Impulzní šum – elektromagnetické rušení
    - Typicky nějaká interference například se silovým vedením, při zapnutí spotřebiče atd
    - Vadí, ale není trvalý
  - De-synchronizace vysílače a přijímače
    - Dojde k de-synchronizaci hodin a vysílač vyšle třetí bit, ale přijímač jej přijme jako druhý nebo čtvrtý
  - Výpadek celého bloku dat
    - Dočasný výpadek spojení – například mokré listí ve WiFi „cestě“

L2: Spolehlivý a nespolehlivý přenos

- !! POZOR !! - Spolehlivost je možné řešit na více vrstvách, zde se bavíme o L2, ale stejně tak jej řeší TCP na L4
- Spolehlivý přenos
  - Typicky z pohledu vyšší vrstvy, které poskytujeme služby
    - Jednoduše řečeno, že se můžu na přenos spolehnout a je mi jedno jak je zařízen
  - Potřebujeme vědět, že
    - Data se přenesla všechna – nic nechybí
    - Data se přenesla správně – všechny 0 a 1 jsou na svém místě
    - Data se přenesla ve správném pořadí – tedy všechny rámce jsou na svém místě
  - Pokud výše uvedené není splněno, potřebujeme to mít možnost nějak napravit
    - Samoopravné kódy, re-transmit
  - Požadavek je to logický, ale pochopitelně drahý, protože když se něco nepovede, musím to nějak řešit
- Nespolehlivý přenos
  - Můžu, ale nemusím vědět, že se něco nepovedlo
  - Můžu, ale nemusím zjištěný problém nějak řešit
    - „Já sice vím, že je problém, ale také vím, že na výsledek pro který se přenos provádí to nebude mít vliv a tedy to neřeším, protože bych zdržoval“
  - Typicky u služeb, kde výpadek části dat nevadí
    - Například hlasové a multimediální přenosy – výpadek několika málo rámce za sebou nebude ve výsledku vůbec patrný

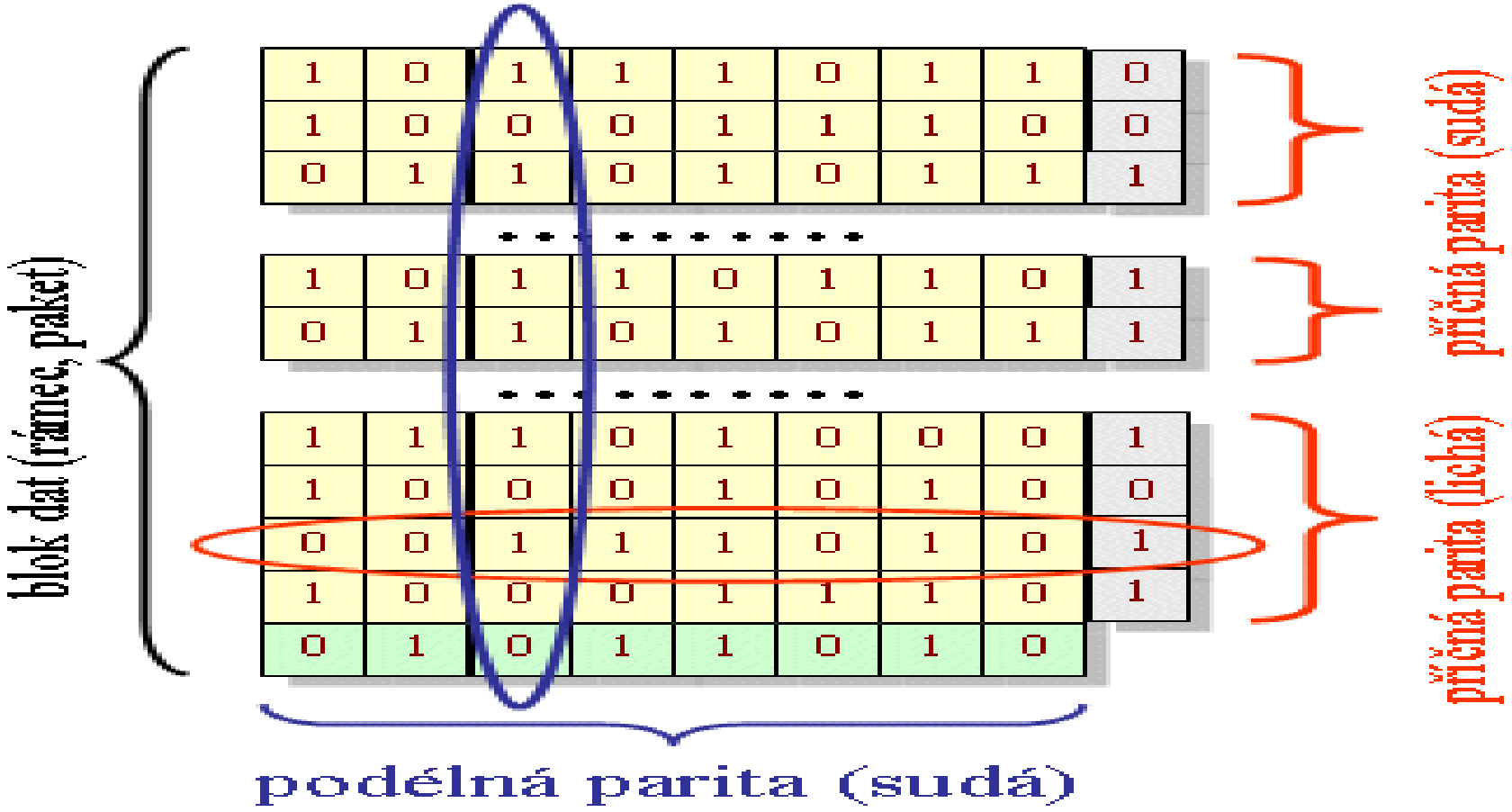
L2: Detekce chyb

- Aby bylo možné realizovat spolehlivý přenos, je nutné se o chybě dozvědět
  - => Detekce chyb
- Detekce chyb je opět možné řešit na více úrovních
  - Detekce výpadku celého bloku / rámce
    - Zjistím tak, že mi nesedí čísla rámců – nějaký chybí nebo přišel ve špatném pořadí
    - V hlavičce rámce musím zavést číslování – ne každý protokol to tak má – viz další přednáška
  - Detekce chyby v jednom rámci / v jednotlivých bitech
    - Zjistím za pomoci přidaných bitů a kontrolních mechanizmů jako jsou:
      - Parita
      - Kontrolní součty / check sumy
      - Cyklické kódy / CRC
    - Výsledek kontrolního mechanizmu je přípojen na konec dat a přenášen společně s nimi s slouží jako podklad pro kontrolu správnosti přenosu
- Opakování / souvislost
  - Podobně jako u arytmického přenosu zde vstupuje do hry využití přenosového kanálu
    - Přenáším data, které přenášet nechci, ale musím => režie přenosu, n = N/M
  - Abych minimalizoval vliv režie, chci co nejdelší blok dat – na L2 rámec, ALE
    - Každý jeden bit se přenese úspěšně s pravděpodobnosti p_1, respektive s chybou $q=1-p_1$
    - Pravděpodobnost úspěšného přenosu N bitů je $P_N = p_1^N$
    - Pokud víme s jakou pravděpodobnosti chceme přenos realizovat a jaké je pravděpodobnost chyby, je maximální délka rámce
      $N = \log_{p1}(P_N)$ – CELÝCH bitů – nezaokrouhlujeme, ale ořezáváme – logicky nemůžeme přenést „půl bit“

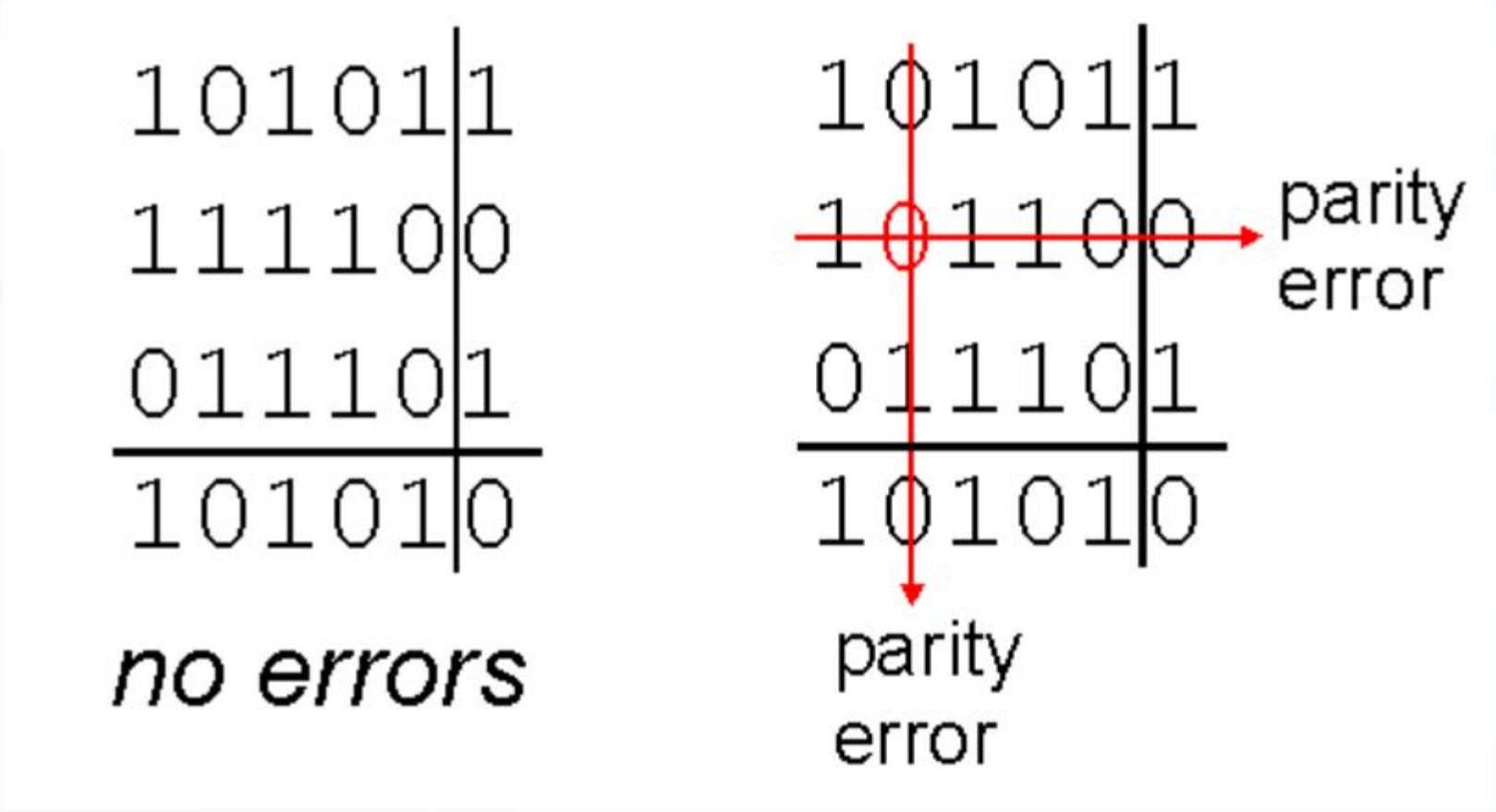
L2: Zabezpečení přenosu: Parita

- O paritě už jsme se zmiňovali u arytmického přenosu, kde byla jako volitelná součást přenosu na L2
  - Jeden z důvodů proč v TCP/IP modulu je L1 a L2 spojené => ne vždy jde přesně rozdělit
- Parita je přidaný 1 bit, kde hodnota je taková aby výsledný počet 1 byl:
  - Sudý: pro 1110|P_S => 1110|1 (doplním 1 na 4 x 1 => sudý počet)
  - Lichý: pro 1110|P_L => 1110|0 (doplním 0, protože v datech mám 3 x 1 => lichý počet)
- Základní, tedy sudá nebo lichá parita, NEMUSÍ odhalit chybu vždy
  - Problém u násobných chyb – pokud se mi změny dva bity, chyba se sečte, parita bude v pořádku, ale data ne => DETEKCE není na 100%
    - Následně se musí řešit i na vyšších vrstvách – dle potřeb protokolu / služby
- Další možností jsou složené parity pro bloky dat
  - Podélná parita – přidám paritní bit pro každý sloupec v bloku
  - Příčná parita - přidám paritní bit pro každý řádek (slovo) v datech
- Podélná i příčná parita mohou být jak sudé tak liché tak jejich kombinace
  - Záleží na autorovi přenosu jaké nastaví parametry
- Existuje i „jedničková“ a „nulová“ parita – tedy paritní bit je vždy 1 nebo 0
  - Tento druh „parity“ nemá žádný zabezpečovací význam

L2: Zabezpečení přenosu: Parita příklad



L2: Zabezpečení přenosu: Parita – detekce chyby - příklad

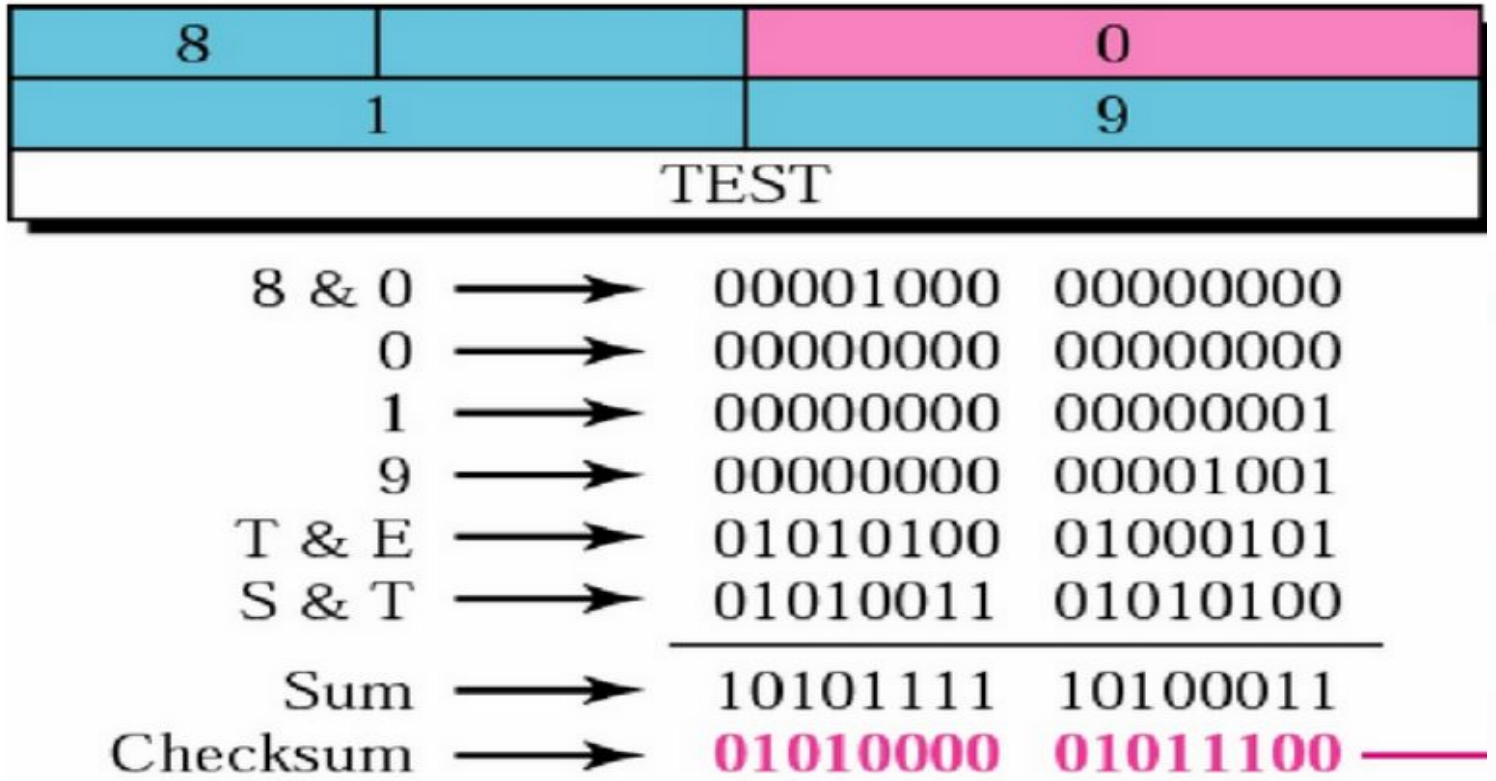


Zdroj: https://www.net.t-labs.tu-berlin.de/teaching/computer_networking/05.02.ht

L2: Zabezpečení přenosu: Kontrolní součet / Checksuma

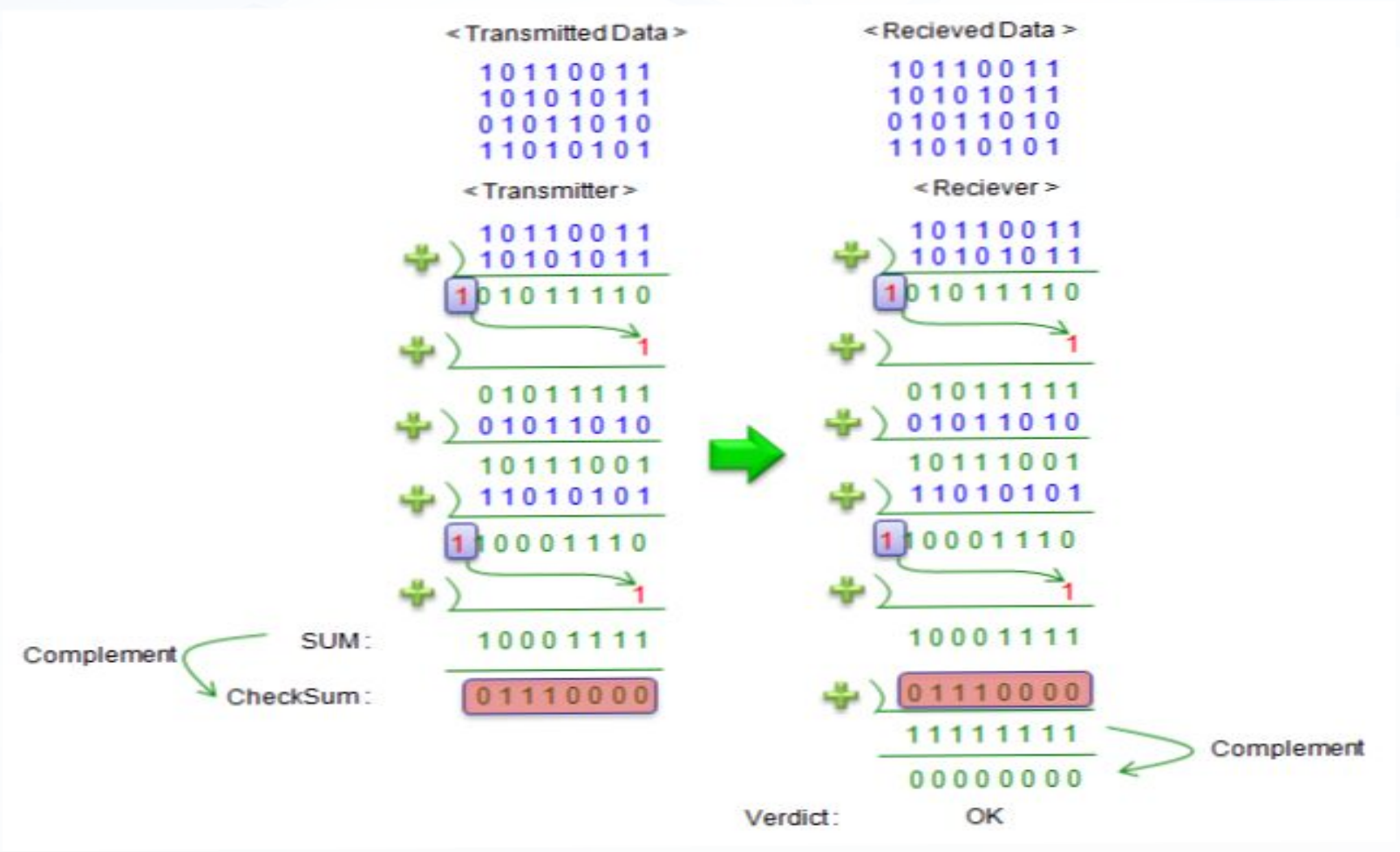
- Jednoduchá parita není příliš účinná, protože neodhalí násobné chyby
  - Ale přesto má smysl, protože se snadno realizuje a nepotřebuje „mnoho“ data navíc
- Podélná a příčná parita už nají účinnost lepší – odhalí více chyb, ALE
  - Spotřebuje více bitů => klesá efektivita využití přenosového kanálu
  - Pro podélnou paritu potřebujeme držet celý blok dat aby bylo možné ji spočítat
    - Paměťové i časově nepříjemné
- Kontrolní součet se snaží řešit výše uvedené problémy
  - Data jsou interpretována jako „slova“ - tedy jednotlivé byty – které mohou být znak nebo číslo
  - Jednotlivá slova, tak jak probíhá přenos se postupně sčítají
    - Mohu dělat průběžně, takže nepotřebuji držet celý blok dat v paměti, ale vždy jen jedno slovo / byty
      - Může jít o sčítání nebo o XOR
  - Výsledný součet se připojí k datům a odešle společně s nimi a na straně příjemce se opět provede kontrolní součet (!! POUZE DAT, kontrolní součet do výpočtu nevstupuje) a porovná s přeneseným kontrolním součtem
    - Přesněji se nepřenáší přímo výsledek, ale jen zbytek po modulu N, kde N je délka slova (pro 1 byte je to 8)
    - Tím, že přenášíme zbytek po modulo N, nebude ten zbytek nikdy delší než N => maximální pevná délka

L2: Zabezpečení přenosu: Kontrolní součet / Checksuma příklad



Zdroj: https://www.net.t-labs.tu-berlin.de/teaching/computer_networking/05.02.htm

L2: Zabezpečení přenosu: Kontrolní součet / Checksuma - příklad



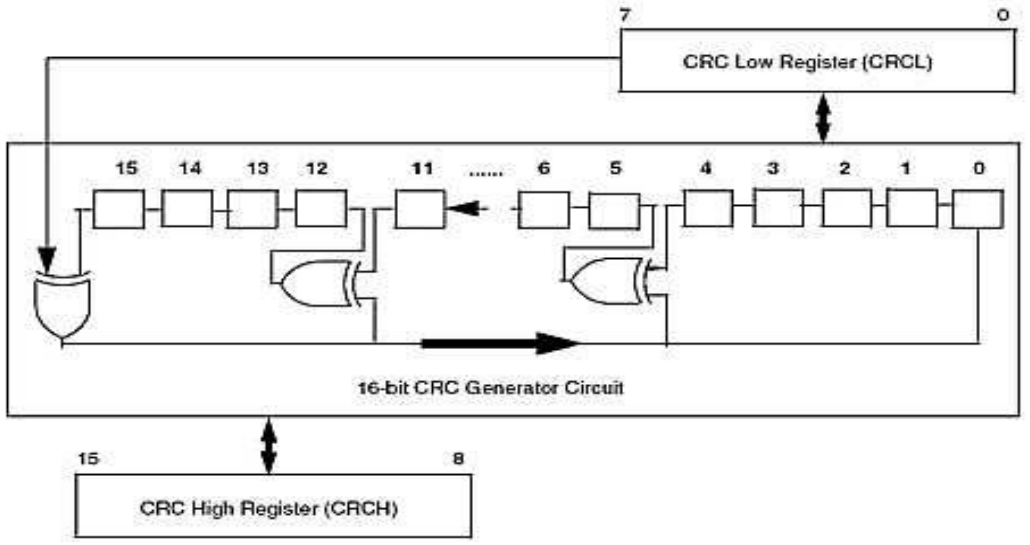
Zdroj: http://www.sharetechnote.com/html/Communication_ErrorCorrection.html

L2: Zabezpečení přenosu: CRC

- CRC - Cyclic Redundancy Check - cyklické zabezpečovací kódy / polynomy
- Ani parita ani kontrolní součet nemají dostatečnou přesnost v detekci chyb
- CRC představuje další možnost v zabezpečení a detekci chyb v přenosu s výrazně lepšími výsledky než parita nebo checksuma
- CRC interpretuje posloupnost dat (0 a 1) - „slovo“ - jako polynom N-tého řádu
  - 101 => $1*x^2+0*x^1+1*x^0$ => x^2+1
- Princip zabezpečení vychází z dělení polynomů:
  - Máme data, která chceme vysílat M(x)
  - Zvolíme zabezpečující polynom G(x) – někdy také označován jako generující polynom
    - Například pro CRC-16: $x^{16}+x^{15}+x^2+1$
  - Provedeme dělení M(x)/G(x) => R(x)
    - Před dělením přidáme k M(x) počet nulových bitů odpovídající stupni zabezpečujícího polynomu -1
    - Kde R(x) není výsledek dělení, ale zbytek po dělení
    - Zbytek po dělení nikdy nebude delší než je řád generujícího polynomu
      - => máme pevnou délku zabezpečení
    - Zabezpečující polynom může být výrazně kratší než přenášená data
      - => máme dobré využití přenosového kanálu
  - Přenášet budeme T(x), kde T(x) = M(x)+R(x) (prosté přidání zbytku po dělení za data)
  - Po přijetí dat na straně přijímače provedeme kontrolní dělení T(x)/G(x)
    - Pokud zbytek po kontrolním dělení není nula => nastala chyba
    - Pokud zbytek po kontrolním dělení je nula => přenos se podařil správně

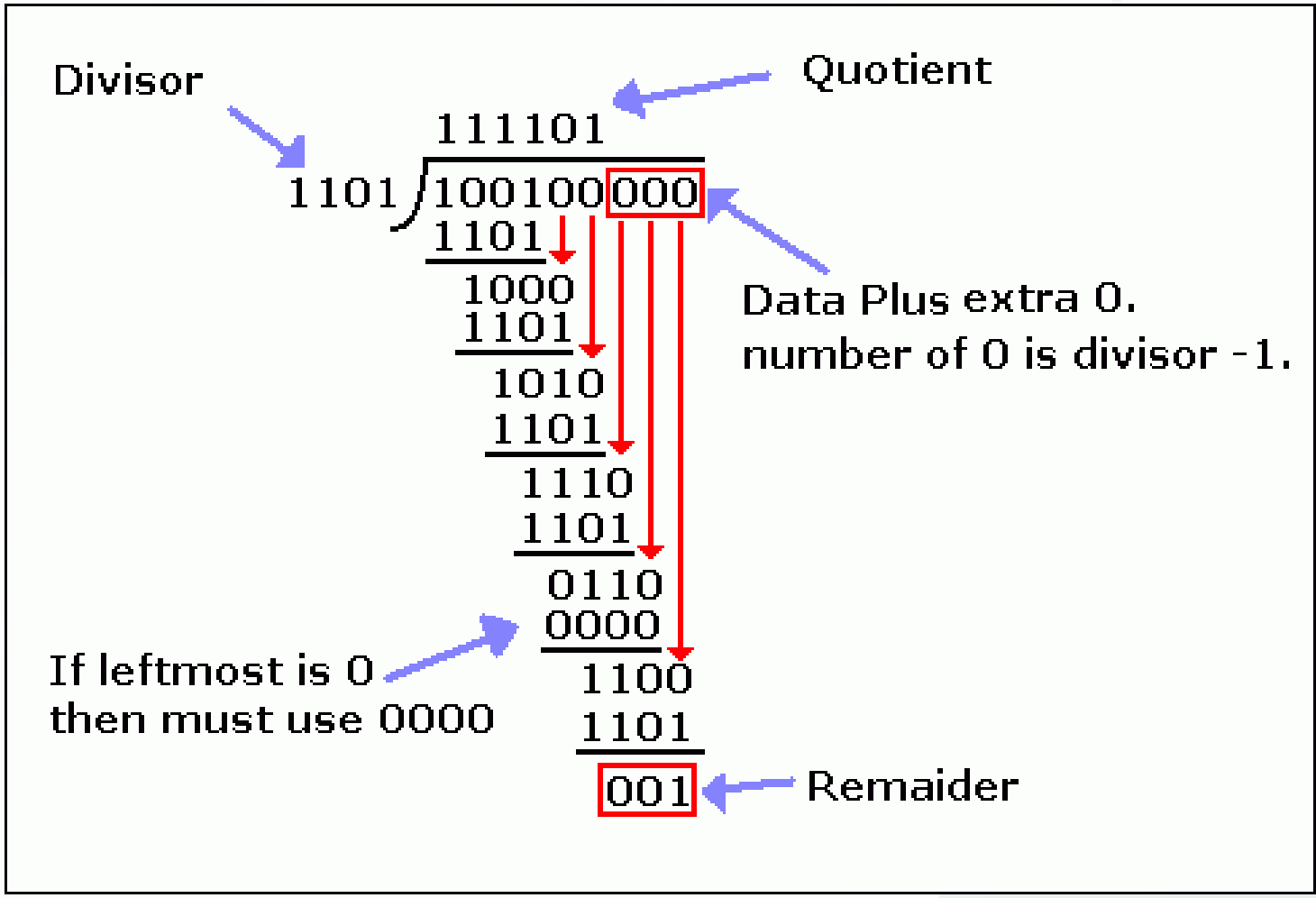
L2: Zabezpečení přenosu: CRC II.

- Schopnost detekce chyb je u CRC velice vysoká
  - Dokáže detekovat všechny shluky chyb o velikosti >N+1 s pravděpodobností 99.99999998% (pro CRC-32 např)
    - Tedy chyby, které jsou delší než je zabezpečující polynom
    - To je výhodné, protože na úrovni rámců je častější shluk chyb než samostatná chyba
- Výpočet CRC je jednoduše realizovatelný na úrovni HW
  - Soustava XOR-hradel a posuvných registrů
- Zásadní a složitá je vhodná volba zabezpečujícího polynomu



L2: Zabezpečení přenosu: CRC příklad výpočtu

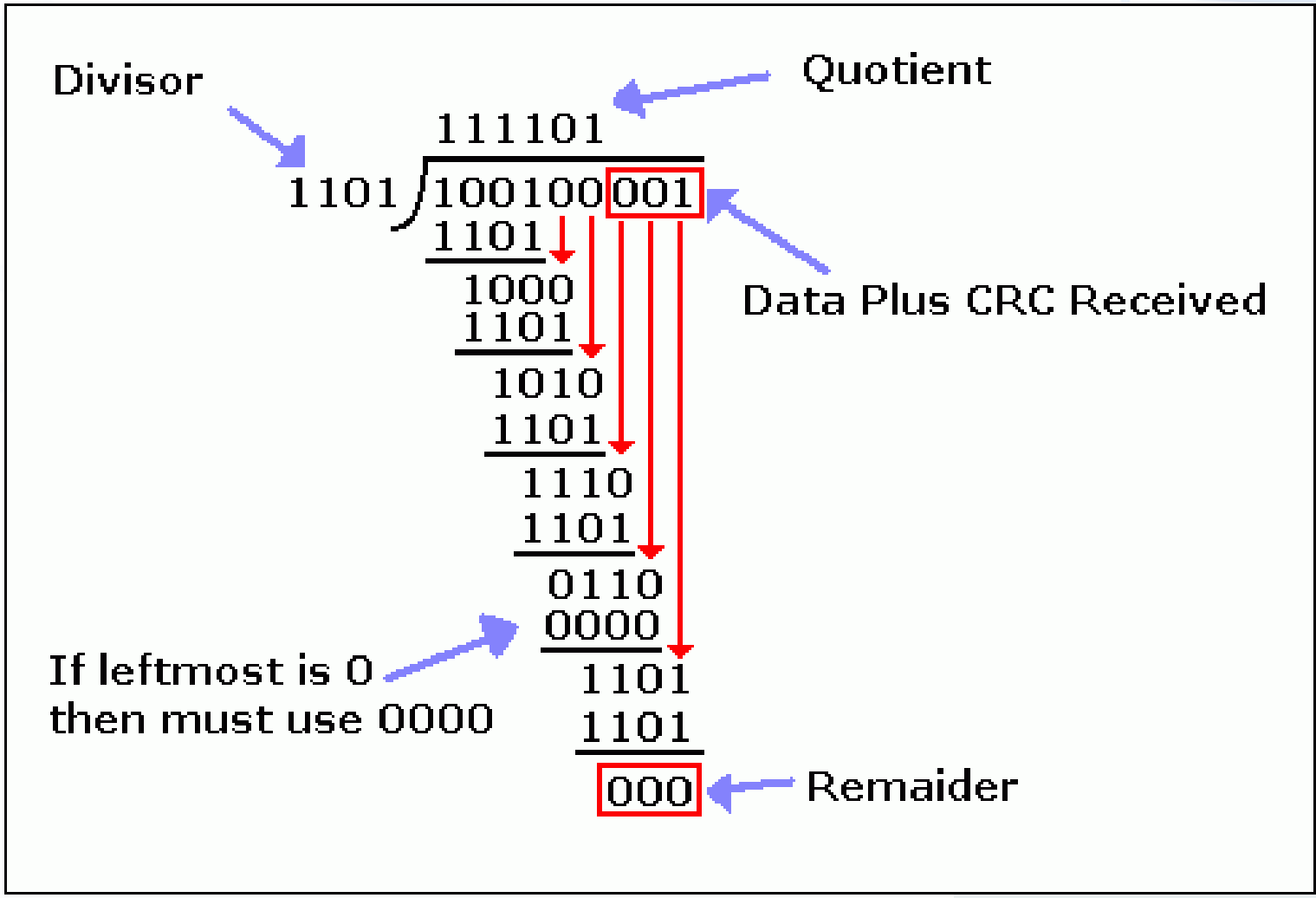
- Máme data M(x)=111101
- Máme zabezpečující polynom
  G(x) = 1101
- Za M(x) přidáme nulové bity
  - Počet „řád“ G(x)-1 = 3
- Provedeme dělení, kde dostaneme zbytek
  R(x) = 001
- Přenášená zpráva bude
  T(x) = 111101|001



Zdroj: http://t4planet.tripod.com/cyclicredundancycheck.htm

L2: Zabezpečení přenosu: CRC příklad kontroly

- Máme zprávu T(x)=111101|001
- Máme zabepečující polynom G(x) = 1101
- Provedeme kontrolní dělení, kde R(x) = 000
  - => Přenos se realizoval správně



Zdroj: http://t4planet.tripod.com/cyclicredundancycheck.htm

L1/L2: Opakování: Využití kapacity přenosového kanálu

- U arytmického přenosu jsme měli 1 paritní bit a start a stop bit a už to negativně ovlivnilo využití kapacity přenosového kanálu, co potom CRC na např 15 bitech ?
- Příklad 1: mějme datový rámec o délce 32 bitů a zabezpečující polynom o délce 15 bitů, jaké bude využití přenosového kanálu ?
  - n = N/M, N = 32 bitů, M = 32 + 15 = 47, n = 32/47 = 0,68 ~ 68% => bída
- Příklad 2: mějme datový rámec o délce 1500 bitů a zabezpečující polynom o délce 15 bitů, jaké bude využití přenosového kanálu ?
  - n = N/M, N = 1500 bitů, M = 1500 + 15 = 1515, n = 1500/1515 = 0,99009901 ~ 99% => lepší

L1/L2: Opakování: Pravděpodobnost úspěšnosti přenosu

- Příklad 3: mějme datový rámec o délce 1500 bitů a zabezpečující polynom o délce 15 bitů, pravděpodobnost chyby v 1 bitu 99.999% - jaká bude pravděpodobnost úspěšného přenosu ?
  - $P_N = p_1^{N,}$, p_1=0.99999, N=1515, $P_N = 0.99999^{1515}$ =0,98496 ~ 98%
- Příklad 4: Jak se změní pravděpodobnost spěšného přenosu, pokud prodloužíme délku datového rámce na 15000 bitů ?
- $P_N = p_1^{N,}$, p_1=0.99999, N=15015, $P_N = 0.99999^{15015}$ =0,86058 ~ 86%

- => nemohu délku rámce zvyšovat bez následků
- => je nutné hledat vhodný kompromis bez využitím kapacity přenosového kanálu a pravděpodobností chyb/úspěšnosti přenosu