Báječný svět počítačových sítí

Část IX. – Zajištění spolehlivosti počítačových sítí

JIŘÍ PETERKA

Některé aplikace vyžadují, aby přenosy dat fungovaly spolehlivě a nedocházelo při nich k chybám ani ztrátám. Jiným aplikacím naopak nějaké poškození dat až tak nevadí a dávají přednost rychlosti, pravidelnosti a malému zpoždění. Když už je ale spolehlivý přenos požadován, jak jej zajistit? Co jsou detekční a samoopravné kódy? Jak funguje potvrzování a jaké má varianty?

Na první pohled by se mohlo zdát, že pokud se někde přenáší nějaká data, mělo by jít o přenos spolehlivý. Tedy takový, který data ani neztratí, ani nedopustí, aby se při přenosu poškodila (změnila). K čemu by pak byl nějaký „nespolehlivý" přenos?

Na druhý pohled už to ale tak jasné není. V praxi totiž mohou existovat (a také reálně existují) takové situace a případy, kterým lépe vyhovuje nespolehlivý přenos. Samozřejmě nikoli takový, který nějak samovolně (sám od sebe) zahazuje či poškozuje přenášená data. „Nespolehlivostí" se myslí to, že když už k nějakému problému dojde (něco se ztratí či poškodí), přenosový mechanismus nepovažuje za svou povinnost postarat se o nápravu. S tím, co se ztratilo, si hlavu neláme a nad tím, co se poškodilo, mávne rukou (zahodí to) – a hlavně se nezdržuje a pokračuje dál, jako kdyby se nic nestalo.

Spolehlivost může vadit

Naopak pokud by se jednalo o spolehlivý přenos, měl by každý, kdo ho zajišťuje, v takové situaci plné ruce práce s nápravou toho, co se stalo. Musel by nejprve zjistit, že vůbec došlo k něčemu nežádoucímu – a pak by si musel vyžádat nový (opakovaný) přenos těch dat, která se ztratila či přišla poškozená. To by ale způsobilo poměrně velké zdržení, které by současně nabouralo průběžné doručování přenášených dat. A právě to může někomu (resp. něčemu) docela vadit.

Představme si třeba, že jde o přenos nějakého živého obrazu a ten je přenášen po vzoru filmového pásu – každé políčko jako jeden samostatný blok (paket či rámec). Příjemce potřebuje dostávat takovéto bloky pravidelně, aby je mohl stejně pravidelně zobrazovat (posouvat filmový pás konstantní rychlostí). Pokud by ale došlo k opakování přenosu nějakého bloku (políčka na filmovém pásu), pravidelnost by se okamžitě narušila a lidské oko by to ihned poznalo. Kolísání rychlosti posuvu filmového pásu lidské oko zaregistruje poměrně spolehlivě. Naproti tomu výpadek či poškození (zkreslení) jednoho políčka na filmovém pásu nemusí lidské oko ani postřehnout. Pak je ale rozumnější oželet nějaký výpadek (ztracené políčko) či zkreslení (poškození políčka) před narušením pravidelnosti přísunu dat. Navíc takový dočasný výpadek či zkreslení lze relativně snadno „zaretušovat" (interpolací z předchozích a následujících snímků), díky inteligenci a výpočetní kapacitě na straně příjemce.

Někdy je lepší to, jindy ono

Celkově tedy lze konstatovat, že existují případy a situace, kdy je výhodnější nespolehlivý přenos, který dává přednost pravidelnosti doručování dat, před spolehlivým přenosem. Například v již popisovaném případě multimediálních přenosů (živého zvuku a obrazu).

Existují ale i jiné důvody preference nespolehlivého přenosu před přenosem spolehlivým. Podstata tohoto důvodu spočívá v tom, že ani spolehlivý přenos není nikdy zcela a stoprocentně spolehlivý. I jemu totiž může „proklouznout" nějaká chybička v přenášených datech, i když pravděpodobnost je zde opravdu velmi malá.

Nicméně mohou existovat i aplikace, kterým „nedostatečná spolehlivost" vadí. Třeba bankovní aplikace, převádějící peníze z účtu na účet, si musí dávat opravdu velký pozor na každou nulu, desetinnou tečku či čárku atd. Podobné aplikace si pak obvykle zajišťují spolehlivost samy, na úrovni, jakou samy potřebují.

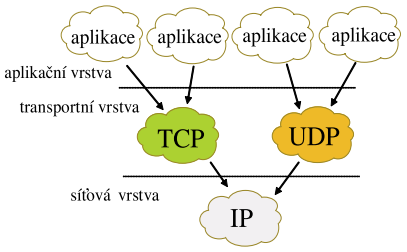
Problém je pak ale v tom, že není rozumné zajišťovat spolehlivost dvakrát či dokonce vícekrát (na různých úrovních, resp. vrstvách „nad sebou"). Ne, že by to nešlo. Jde to, ale je to nevýhodné. Zajištění spolehlivosti totiž přináší nenulovou režii, a to jak z hlediska spotřeby výpočetní kapacity (je nutné zjišťovat, jestli k chybě došlo), tak především z hlediska spotřeby přenosové kapacity a přenosového zpoždění (na opakování přenosu poškozených dat). Pokud se spolehlivost zajišťuje na více vrstvách, režie s tím spojená se v lepším případě pouze sčítá, v horším násobí či kombinuje ještě nevýhodněji.

Z hlediska režie je jednoznačně výhodnější, aby se spolehlivost zajišťovala jen jednou, a to tam, kde je skutečně zapotřebí. „Pod tím" (tedy na nižších úrovních) by se spolehlivost nezajišťovala a přenosy by fungovaly nespolehlivě.

Asi nejlépe to je vyřešené v rámci rodiny protokolů TCP/IP, kde základní přenosový protokol síťové vrstvy (protokol IP) funguje nespolehlivě. Nad ním, na úrovni transportní vrstvy, pak existují dva transportní protokoly:

- spolehlivý protokol TCP, Transmission Control Protocol (zajišťující spolehlivost),
- nespolehlivý protokol UDP, User Datagram Protocol (nezajišťující spolehlivost).

Jednotlivé aplikace si pak mohou svobodně vybrat, který z obou transportních protokolů využijí. Pokud si spolehlivost chtějí zajistit samy nebo o ni naopak nestojí, vybírají si protokol UDP. Pokud o spolehlivost přenosů stojí a nechtějí si ji zajišťovat samy, volí protokol TCP. Představu ukazuje následující obrázek.

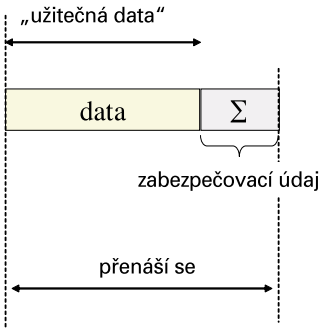


Aplikace v TCP/IP si samy volí mezi dvěma vzájemně alternativními transportními protokoly, TCP a UDP.

Když už tušíme, jak je to se spolehlivostí přenosu dat – je žádoucí jen někdy, nikoli vždy – pojďme si nyní říci, jak se jí vlastně dosahuje. Nebo spíše: jak se spolehlivost přenosu zvyšuje (když už víme, že není nikdy absolutní). I tento úkol si musíme rozdělit na dvě samostatné části, abychom se v tom vyznali:

- na detekci chyb,
- na nápravu detekovaných chyb.

Detekcí chyb se rozumí již samotné zjištění toho, že se „něco stalo" – došlo k nějaké nestandardní situaci, konkrétně k nějaké chybě v přenášených datech. Správně by sem měla patřit úplná ztráta přenášeného bloku dat (paketu, rámce), ale k tomu se dostaneme záhy. Nyní se soustředíme na to, jak dokáže příjemce rozpoznat, že v přijatých datech je nějaká chyba.



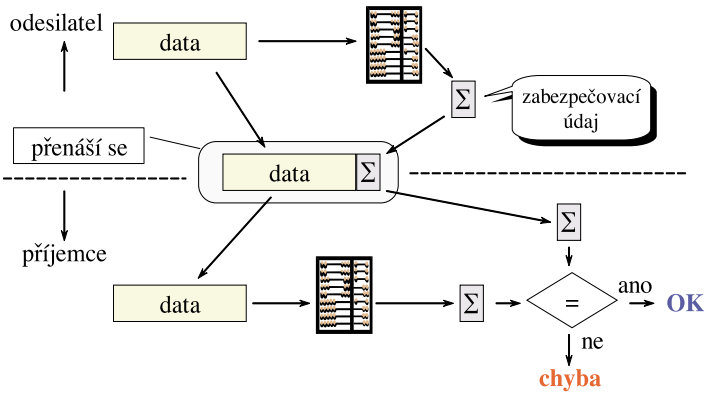
Zabezpečovací údaj

Aby příjemce mohl rozpoznat, zda přijatá data jsou či nejsou v pořádku, musí k nim odesilatel „přibalit" určitý dodatečný údaj, kterému se obecně říká „zabezpečovací údaj". Odesilatel tento zabezpečovací údaj určitým způsobem vypočítá z dat, která mají být odeslána. Pak je přidá k těmto (užitečným) datům a přenese k příjemci výsledný celek, viz obrázek.

Představa přenášeného bloku dat se zabezpečovacím údajem.

Příjemce postupuje obdobně: vezme přijatá (užitečná) data, sám z nich vypočítá zabezpečovací údaj (přesně stejným postupem, jako to dělal odesilatel) a výsledek porovná se zabezpečovacím údajem, který mu přišel od odesilatele spolu s daty. Pokud se oba zabezpečovací údaje liší, může z toho příjemce usuzovat, že se něco po cestě poškodilo a vyžádat si opakované zaslání stejných dat. Samozřejmě se mohlo stát i to, že se užitečná data přenesla v pořádku a poškozen byl pouze přenášený zabezpečovací údaj – ale toto příjemce není schopen rozlišit.

V opačném případě, tedy pokud se oba zabezpečovací údaje shodují, příjemce předpokládá, že přenos proběhl bez chyb a pokračuje dál. Opět ale nemá stoprocentní jistotu, že tomu tak skutečně je. Představu ukazuje další obrázek.



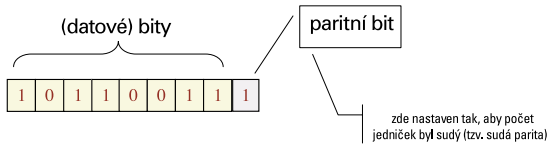
Představa použití zabezpečovacího údaje při přenosu dat.

Mechanismy detekce chyb

Důvod, proč příjemce nemůže mít stoprocentní jistotu o bezchybnosti přenosu, a to ani v případě shody obou zabezpečovacích údajů, je prostý – stoprocentní nejsou již samotné mechanismy detekce, podle nichž je konstruován zabezpečovací údaj a podle nichž se chyby detekují. I těmto mechanismům může občas nějaká chybička uniknout.

Jaké mechanismy se ale používají pro detekci chyb a jak tyto mechanismy vlastně fungují?

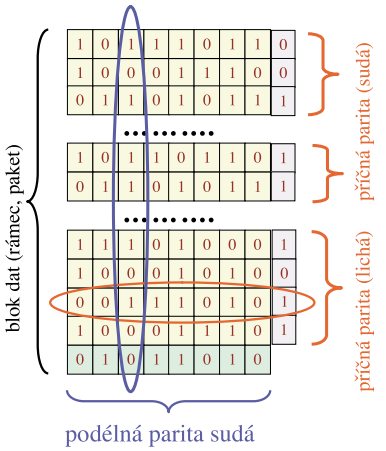
Mezi ty nejjednodušší, ale bohužel také nejméně účinné, patří tzv. parita. Její princip je jednoduchý: bity, které mají být přeneseny, se doplní o další bit (tzv. paritní bit), který se nastaví tak, aby celkový počet bitů byl buď sudý (pak jde o tzv. sudou paritu), nebo naopak lichý (tzv. lichá parita). Představu (sudé parity) ukazuje opět další obrázek.



▲ Představa sudé parity.

Jistě není těžké nahlédnout, že parita dokáže odhalit (detekovat) chybu v lichém počtu bitů. Ovšem chyby v sudém počtu bitů se z pohledu parity vzájemně ruší a nejsou tedy odhaleny. V praxi bývají paritou (paritním bitem) opatřovány relativně malé skupiny bitů, kde je pravděpodobnost výskytu chyby poměrně malá. Tedy například jednotlivé byty či slova (dvojice bytů, čtveřice atd.). Příkladem může být asynchronní (správně arytmický) přenos, o kterém jsme hovořili v minulém dílu tohoto seriálu.

V případě přenosu větších datových bloků (rámců, paketů) je dnes použití parity spíše vzácností, a to kvůli její relativně nízké účinnosti. Ale když už je i zde parita použita, může být použita dvěma různými způsoby (případně současně):



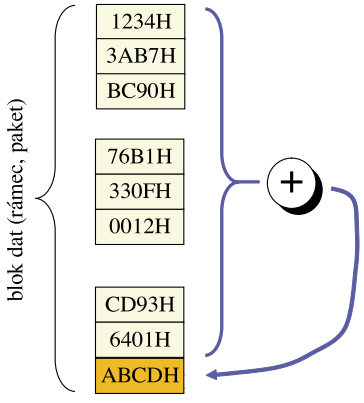
▲ Podélná a příčná parita.

- Jako tzv. příčná parita, kdy je každý byte (či slovo) v přenášeném bloku opatřen vlastním paritním bitem.
- Jako tzv. podélná parita, kdy jsou jedním paritním bitem zabezpečeny „stejnolehlé" bity ve všech bytech, resp. slovech přenášeného bloku.

Vše názorně ukazuje obrázek.

Na příkladu parity si také můžeme předvést jeden významný aspekt, který souvisí se zajištěním spolehlivosti přenosu. Obě varianty parity totiž umožňují detekovat, ve které části přenášeného bloku k chybě došlo. V případě příčné parity se příjemce dozví, ve kterém bytu (případně slově) přenášeného bloku k chybě došlo, zatímco v případě podélné parity se dozví, ve které skupině „stejnolehlých" bitů se tak stalo. Je ale taková informace příjemci vůbec k něčemu?

Podstatné je, jak bude příjemce reagovat na výskyt nějaké (jakékoli) chyby v přijatém bloku. V případě nespolehlivého přenosu zahodí celý blok a už se o něj dále nestará. V případě spolehlivého přenosu si nechá celý blok poslat znovu. Takže k čemu mu je informace o tom, ve které části přenášeného bloku k chybě došlo? Není mu k ničemu, protože ji nedokáže využít. Stačí mu podstatně méně podrobná informace o tom, že někde (kdekoli) v rámci bloku došlo k nějaké chybě (nebo chybám).



▲ Představa kontrolního součtu.

S touto skutečností počítají i další mechanizmy, které se již nesnaží „lokalizovat" chybu a soustřeďují se na lepší (přesnější) detekci jejího výskytu, kdekoli v bloku přenášených dat.

Jedním takovým mechanizmem je kontrolní součet. Funguje tak, že přenášený blok se chápe jako lineární posloupnost bytů (resp. slov, tedy dvojic či čtveřic bytů). Každý prvek této posloupnosti se interpretuje jako číslo, tato čísla se sečtou. Jako zabezpečovací údaj se pak použije výsledný součet (ořezaný na stejný počet bitů, jaký mají jednotlivé sčítance). Představu ilustruje obrázek vlevo dole.

Kontrolní součty dokáží detekovat více chyb než parita. Je to dáno už jen tím, že dvojice chyb se většinou vzájemně neruší, jako je tomu u parity. Na druhou stranu stále mohou existovat kombinace chyb, které se navzájem vyruší a dávají stejný kontrolní součet jako nepoškozený blok. Takže i kontrolním součtům mohou některé chyby uniknout.

Nejspolehlivější a v praxi nejpoužívanější je detekce chyb pomocí tzv. cyklických kódů, známějších spíše podle anglické zkratky CRC (Cyclic Redundancy Check). Jejich účinnost je přímo vynikající. Dokáží detekovat všechny shluky chyb s lichým počtem bitů, dále všechny shluky chyb do velikosti n bitů (kde n je počet bitů zabezpečovacího údaje). Všechny větší shluky chyb pak CRC dokáží detekovat s pravděpodobností 99,99999998 %. Výhodou je poměrně snadná implementace a praktický výpočet zabezpečovacího údaje. Stačí k tomu poměrně jednoduchý obvod, který rychle zajistí vše potřebné. Kde je potom nějaký háček?

Vlastně nikde. Snad jenom v tom, že v pozadí za celou touto metodou detekce chyb stojí silná matematická teorie. Ta také poskytuje návod na to, jak zabezpečovací údaj konkrétně počítat. Abychom se s tím mohli seznámit, museli bychom zabrousit opravdu hodně hluboko do algebry, na což zde rozhodně nemáme prostor. Proto se ani nebudeme snažit nějak zjednodušeně popsat, jak se vlastně takové „CRCéčko" počítá. Spokojíme se jen s konstatováním, že způsob využití je stejný jako u kontrolních součtů – zabezpečovací údaj se připojí k bloku, přenese se spolu s ním a na straně příjemce se počítá znovu a porovnává s přeneseným. Výsledkem je konstatování, že v bloku jako takovém je (resp. není) chyba. Opět, na rozdíl od parity, není patrné, v které části bloku k případné chybě došlo. Rozdíl oproti kontrolním součtům spočívá v podstatně vyšší spolehlivosti, s jakou je možná chyba detekována. Proto se také CRC v praxi používá velmi často.

Mechanizmy pro opravu chyb

Výše popisované mechanizmy (parita, kontrolní součty i CRC) jsou pouze detekčními prostředky. Jsou určené pouze k tomu, aby detekovaly výskyt jedné či více chyb. Zjišťovat počet chyb vlastně ani nemusí, protože se od nich očekávají pouze dvě možné odpovědi: „celý blok je v pořádku" a „blok není v pořádku". Další pokračování se totiž vždy týká celého bloku jako takového: buď je celý zahozen (v případě nespolehlivého přenosu), nebo je vyžádán opakovaný přenos celého bloku (v případě spolehlivého přenosu).

Někdy ale je zapotřebí zajistit spolehlivý přenos, resp. zvýšit jeho spolehlivost, a není možné opakovat přenos poškozeného bloku dat. Třeba proto, že vůbec neexistuje možnost přenosu v opačném směru. Nebo proto, že takový přenos by neúnosně narušil pravidelnost doručování dat (viz výše). Co dělat v takovém případě?

I zde existuje určité řešení, spočívající v použití takových zabezpečovacích údajů, které dokáží chyby nejen detekovat, ale také je opravovat (u příjemce). V praxi se hovoří o tzv. samoopravných kódech (nejčastěji jsou v této roli používány tzv. Hammingovy kódy). V angličtině se zase lze setkat se zkratkou FEC, Forward Error Control, ve smyslu: dopředná ochrana proti chybám. Proč se ale tato možnost používá spíše vzácně a přednost se dává opakovanému přenosu poškozených dat?

Problém je v tom, že samoopravné kódy mají příliš velkou režii. Aby dokázaly opravit byť jen „malé" chyby, musí přenést mnohem více dat. Princip jejich fungování si lze představit tak, že místo jedné „sady" dat jich přenáší hned několik (dvakrát, třikrát atd., ovšem stejná data) a z nich se u příjemce vybírá tu sada, která se přenesla bez poškození. V praxi to funguje poněkud efektivněji (dat se nepřenáší n× více), ale zase ne o tolik. Proto je celá tato varianta zajištění (zvýšení) spolehlivosti používána jen tam, kde opakované zaslání poškozených dat nepřipadá v úvahu.

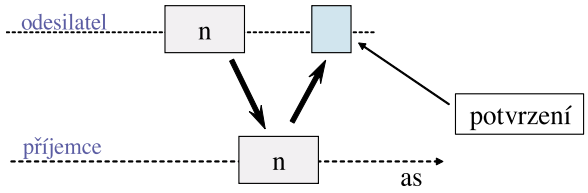
Potvrzování

Pojďme nyní zpět k tomu, jak se spolehlivý přenos zajišťuje nejčastěji – přes opakovaný přenos poškozených dat. Základní podmínkou je to, aby se ode-

silatel vůbec dozvěděl, zda příslušný datový blok byl či nebyl přenesen v pořádku. Z toho si pak již odvodí, zda přenos bloku opakovat či nikoli.

Nutnou podmínkou k tomu, aby se odesilatel cokoli dozvěděl od příjemce, je existence zpětné vazby, resp. možnost přenosu „v opačném směru" (směrem od příjemce dat k jejich odesilateli). V praxi to znamená, že mezi oběma stranami musí existovat obousměrný přenosový kanál, resp. okruh. Ne nutně plně duplexní, ale alespoň poloduplexní. Přes takový obousměrný kanál pak mohou obě strany vést určitý dialog, využívající mechanismus tzv. potvrzování.

Potvrzování (anglicky acknowledgement) funguje přesně tak, jak napovídá jeho název: potvrzuje, že určitý blok dat byl přenesen. Jde tedy o zprávu, kterou generuje příjemce datového bloku a přijímá ji odesilatel tohoto bloku. Cestuje tedy v opačném směru, než v jakém je přenášen ten datový blok, který potvrzuje (od příjemce k odesilateli, viz obrázek).



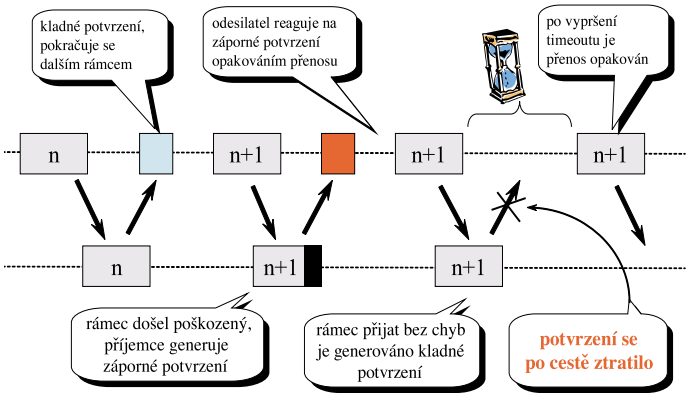
▲ Princip potvrzování.

Jednotlivé potvrzování

Důležité je, že potvrzení může být jak kladné (potvrzující bezchybný přenos), nebo naopak záporné (signalizující chybně přenesený blok). Podstatné je také to, jak odesilatel s potvrzením nakládá.

Jedna základní možnost je taková, že když odesilatel odešle nějaký blok dat, nejprve čeká na odezvu druhé strany. Pokud dostane kladné potvrzení, pokračuje odesláním dalšího bloku (a znovu čeká na potvrzení atd.). Pokud dostane záporné potvrzení, odešle stejný blok znovu. Kromě toho ale musí počítat ještě se dvěma dalšími možnostmi:

- Mohl by se ztratit celý přenášený blok, takže příjemce ani neví, že by měl něco potvrzovat.
- Ztratilo se samotné potvrzení, které příjemce vygeneroval.



▲ Jednotlivé potvrzování.

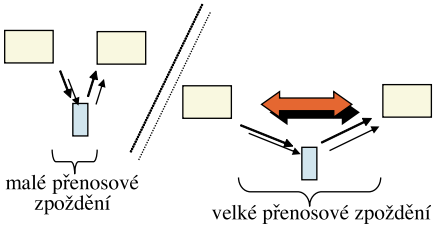
V obou případech odesilatel musí určitou dobu čekat (po určitý časový limit, v angličtině timeout). Když do uplynutí této doby nedostane žádné potvrzení, interpretuje to stejně jako potvrzení záporné a přenos bloku opakuje. Celý právě popsaný postup je označován jako tzv. jednotlivé potvrzování (v angličtině Stop&Wait ARQ).

Snad není těžké pochopit, že velikost časového limitu musí být volena velmi pečlivě. Pokud by byla příliš krátká, odesilatel by mohl „předčasně zpanikařit" (a opakovat přenos dříve, než mu potvrzení přijde). Naopak po-

kud byl časový limit příliš velký, odesilatel by čekal zbytečně dlouho a přenos by se zbytečně zpomaloval.

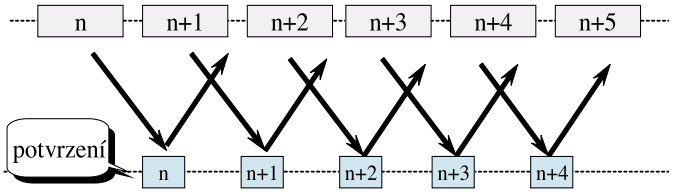
Kontinuální potvrzování

Jednotlivé potvrzování je vhodné pouze pro takové sítě, ve kterých „cesta tam a zpět", od odesilatele k příjemci, netrvá dlouho. Tedy tam, kde je malé tzv. přenosové zpoždění. Týká se to hlavně lokálních sítí (LAN). V rozlehlých sítích (sítích WAN) je přenosové zpoždění relativně velké – a kvůli tomu neúměrně narůstají prodlevy, způsobené čekáním na potvrzení každého jednotlivého bloku.



▲ Vliv přenosového zpoždění při jednotlivém potvrzování.

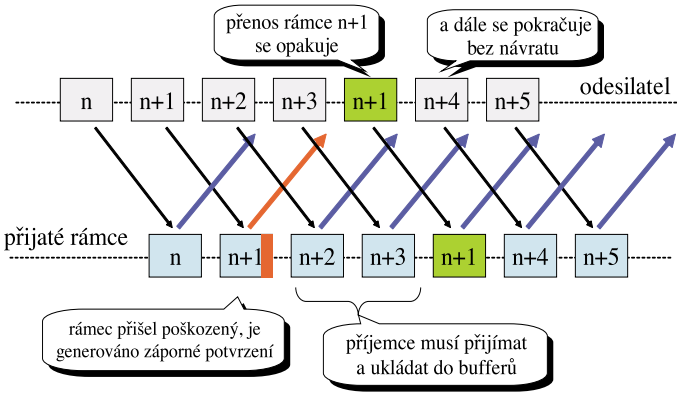
V prostředí rozlehlých sítí je proto výhodnější jiná strategie, označovaná jako kontinuální potvrzování. Od jednotlivého potvrzování se liší v tom, když odesilatel odešle nějaký blok, nečeká na jeho potvrzení, ale pokračuje odesláním dalšího bloku. Posílá tedy jednotlivé datové bloky „jeden za druhým" a teprve zpětně přijímá potvrzení o tom, jak byly tyto bloky doručeny.



▲ Představa kontinuálního potvrzování.

Jak je tomu ale v případě, že odesilateli přijde záporné potvrzení? Nebo pokud se ani po uplynutí časového limitu (timeoutu) nedočká žádného potvrzení a musí vše interpretovat jako záporné potvrzení? Zde existují dvě možné varianty, jak odesilatel zareaguje.

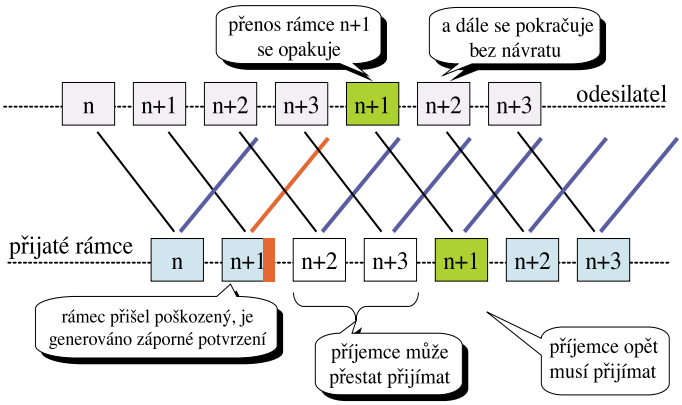
První varianta je označována jako tzv. selektivní opakování. Spočívá v tom, že odesilatel zopakuje (znovu odešle) právě a pouze ten blok, který se poškodil či ztratil (ke němuž se vztahuje záporné potvrzení) a dále pokračuje, jako kdyby se nic nestalo.



▲ Představa kontinuálního potvrzování se selektivním opakováním.

Nevýhodou selektivního opakování je vyšší náročnost na příjemce. Pokud totiž přijme nějaký poškozený blok (nebo jej nepřijme vůbec), ještě nějakou dobu poté musí přijímat následující bloky, které ale nemůže zpracovávat. Místo toho je musí někam ukládat a čekat, až dostane původně poškozený či ztracený blok znovu. Teprve pak může začít zpracovávat i následující, již přijaté bloky.

Uvedený problém řeší druhá varianta kontinuálního potvrzování, označovaná jako opakování s návratem. Spočívá v tom, že když odesilatel dostane záporné potvrzení (nebo mu vyprší časový limit), znovu odešle příslušný blok a poté pokračuje těmi bloky, které po něm následují (bez ohledu na to, že některé z nich mohly být odeslány v mezidobí). Jinými slovy: vrátí se k poškozenému bloku a dál pokračuje od něj.



▲ Představa kontinuálního potvrzování s návratem.

I varianta „s návratem" ovšem má své nevýhody. Je sice šetrnější vůči příjemci, ale na druhé straně může zbytečně plýtvat přenosovou kapacitou. Může totiž znovu přenášet bloky, které se již jednou přenesly úspěšně (bez chyb).

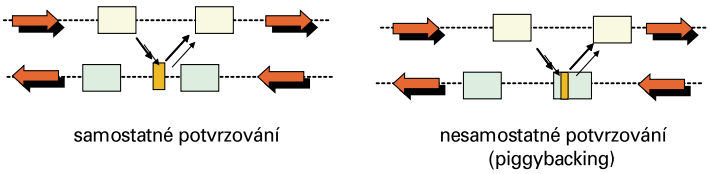
Samostatné a nesamostatné potvrzování

Na závěr se ještě zastavme u toho, jakou konkrétní podobu mohou mít potvrzení, zasílaná od příjemce směrem k odesilateli. Jednou možností je, že potvrzení jsou zcela samostatná, neboli jde o principiálně stejné bloky, v jakých se přenáší i samotná data. Mají tedy svou vlastní hlavičku, která určitým způsobem zvyšuje režii na přenos.

Snaha eliminovat tuto režii vedla ke vzniku tzv. nesamostatného potvrzování (anglicky piggybacking). To spočívá v tom, že příjemce původních dat čeká s jejich potvrzením, dokud „v protisměru" necestuje nějaký jiný datový blok, z nějakého jiného přenosu – a do něj jen „přidá" své potvrzení. Využívá tedy toho, že zmíněný blok „v protisměru", patřící nějakému jinému přenosu, již tak jako tak má svou hlavičku a další náležitosti a tudíž je přírůstek režie na zajištění potvrzení minimální.

I nesamostatné potvrzování však má svá úskalí. Zejména to, že příjemce nemůže čekat příliš dlouho na to, až bude „v protisměru" přenášen nějaký jiný datový blok, do něhož by své potvrzení vložil. Odesilatel o tom totiž neví a mohl by mu vypršet časový limit (timeout). V praxi se příjemce snaží o nesamostatné potvrzení jen chvíli, pokud se mu nepodaří, přechází k samostatnému potvrzení (tedy odešle své potvrzení jako samostatný blok).

5 0624/BAM ❑



▲ Představa samostatného a nesamostatného potvrzování.