Számítógépes hálózatok (4.rész)

Adatkapcsolati réteg, keretezés, forgalomirányítás és szabályozás

Belekezdünk az adatkapcsolati réteggel történő mélyebb ismerkedésbe. Lesz szó keretekről, forgalomszabályozásról de még hibajavításról is.

Immár hetedik alkalommal jelentkezik számítógép-hálózatokat boncolgató sorozatunk, amelyet a gyakorlati jelentőséggel nem igazán bíró, ám az elmélet megértéséhez sokat segítő OSI modellre építünk. Idáig sok témát érintettünk már, de részletesen csak azzal a fizikai réteggel foglalkoztunk, amely lehetővé teszi, hogy két „szomszédos" számítógép információkat cseréljen. Ezek az információk bitsorozatok, amelyeket a hálózati réteg vagy elektromágneses hullámok vagy pedig a vezeték bizonyos fizikai jellemzőinek megváltoztatásával továbbít. Miután a hálózati réteg „ráhelyezte" a bitsorozatot az átviteli csatornára, feladata végetér. Ezzel azonban csak félmunkát végzünk, a „szomszédos" számítógépek közötti adatcsere még sincs megoldva. A fizikai réteg ugyanis semmilyen módon nem garantálja azt, hogy a neki átadott bitsorozat sértetlenül megérkezzék (már ha egyáltalán megérkezik) a célhoz. Az átviteli csatornák pedig nem tökéletesek, könnyen előfordulhat, hogy egy-egy bit elveszik útközben, de az is, hogy több érkezik meg, mint amennyit elküldtünk. Ha esetleg a megérkezett bitek számban megegyeznek az elküldöttekkel, akkor pedig értékeikben különbözhetnek.

A hálózati réteg – amely már azon fáradozik, hogy a hálózatban ne csak az egymással közvetlenül összekapcsolt gépek kommunikálhassanak, hanem a hálózat bármely pontjáról bármely pontjára tudjunk csomagot küldeni – helyes működésének alapvető előfeltétele, hogy az adatátvitel többnyire hibamentes legyen, illetve hogy ha mégis hiba történik, akkor az érzékelhető legyen. Ezért van szükség az adatkapcsolati rétegre, amelynek legfontosabb feladata, hogy felismerje az átviteli közeg tökéletlenségéből adódó hibákat, amelyeket ezután lehetőség szerint megpróbál kijavítani.

Keretezés

Az imént említett feladatra a legcélszerűbb megoldás az, ha a bitfolyamot az adatkapcsolati réteg különálló részekre, úgynevezett keretekre(frames) tördeli, és ezekhez hozzácsatol egy úgynevezett ellenőrző összeget (checksum). Az ellenőrző összeg alapján a legtöbb esetben ki lehet szűrni az átvitelkor keletkezett hibákat. A keret megérkezésekor ezt a számot ismét kiszámolja a rendszer, és ha az nem egyezik meg az eredetivel, akkor átviteli hiba történt, amit jelezni kell a küldő felé.

Ez első hallásra nem tűnik valami nagy kunsztnak, de mindjárt kiderül, hogy egyáltalán nem valami triviális feladat. A dolgot ugyanis megnehezíti, hogy a célállomásnak el kell tudnia különíteni egymástól a beérkező kereteket, azaz tudnia kell, hogy egy keret mikor ér véget.

Erre a problémára a legegyszerűbb megoldás az lehet, hogy amikor a forrás elküldte az első keretet, egy meghatározott ideig beszünteti az adást, azaz szünetet tart. Mikor ez a kis idő lejár, elindítja a következő keretet. A hálózatokról azonban tudnunk kell, hogy minden olyan feladat, ahol az időzítés szerepet játszik, a legtöbb esetben kivitelezhetetlen. Ezek a szünetek ugyanis megnyúlhatnak, esetleg lerövidülhetnek, szélsőséges esetekben el is tűnhetnek. A fogadó így képtelen lesz teljes bizonyossággal felismerni a kerethatárokat, ezért más megoldást kell kitalálni.

Egyik alternatíva lehet a keretszámlálás néven elhíresült módszer, amikor minden keret a keretben található karakterek számával kezdődik. Ha például a keret első karaktere egy 8-as, akkor a fogadó tudni fogja, hogy a nyolcadik karakter után érkező bitek már egy új kerethez tartoznak. Ezzel a megoldással az a probléma, hogy a kommunikációban részt vevő két fél nagyon könnyen kieshet a szinkronból. Vegyük például azt az esetet, amikor egy átviteli hiba megrongálja a keret első karakterét, és így a célhoz nem egy 8-as, hanem egy 12-es érkezik. Ennek következtében a következő tizenkét karaktert ehhez a kerethez sorolja, miközben az utolsó négy karakter már egy másik keretnek a része. Persze ez a hiba egyből kibukik az ellenőrző szám alapján elvégzett hibakereséskor. Ez azonban nem sokat számít, a fogadó ugyanis ekkor sem fogja tudni, hol kezdődött a következő keret. Az sem segít, ha a forrás megismétli a hibás keretet, hiszen a cél nem tudja, hány karaktert lépjen vissza ahhoz, hogy a kicserélendő rossz keret valódi kezdetére érkezzen. Ez a módszer sem igazán nyerő tehát.

A keretszámlálás legnagyobb buktatója, hogy nem nyújt lehetőséget a szinkron újbóli felvételére, azaz arra, hogy a vevő valamilyen módon ismét képben legyen a kerethatárokat illetően.

A következő ismertetendő módszer erre már lehetőséget nyújt. Az alapötlet, hogy a keretek kezdetét illetve végét egy-egy speciális karaktersorozat jelölje. A keret kezdetét jelölje például a DLE STX nevű (DLE: ***Date Link Escape***, STX: ***Start TeXt***), a végét pedig a DLE ETX (ETX: ***End TeXt***) nevű két karakterből álló sorozat. Ha mondjuk egy átviteli hiba folytán valamelyik kerethatárt jelölő karakter megsérül, ak-

kor sem menthetetlen a helyzet. A cél ismét szinkronba kerülhet a forrással, ha figyeli a DLE STX és DLE ETX karaktereket.
Ez a módszer csak addig működőképes, amíg nem próbálunk bináris adatokat küldeni (például programkódot, képet, videót, lebegőpontos számot, stb.). Ezek az adatok ugyanis könnyen tartalmazhatnak DLE karaktert, amely speciális jelentéssel bír a célállomás számára, és így megzavarhatja a kommunikációt. A megoldás az, hogy valamiképpen jelölni kell, hogy az adott DLE karakter csak egy, az adatban szereplő karakter, vagy valóban a kerethatár kezdetét jelzi. A leggyakoribb megoldás ilyenkor az, hogy az adatkapcsolati protokoll a szövegben „véletlenül" előforduló DLE karaktereket mindig megduplázza (kétszer küldi el). Ez a módszer már ígéretesnek tűnik, csak az a baj, hogy szorosan összefügg a 8 bites ASCII karakterkódolással. Igaz, hogy ez egy nagyon elterjedt kódolás (főleg a PC-s világban), azonban mégsem várhatjuk el, hogy minden számítógép 8 biten, ezen belül az ASCII kódolás segítségével képezze le a karaktereket. Ha azt szeretnénk, hogy hálózatunknak mindenféle színű és szagú számítógépek is tagjai lehessenek, akkor egy olyan keretező algoritmusra van szükség, amely független a karakterek méretétől és kódolásától.
A megoldás hasonló lesz az előbbihez, azzal a különbséggel, hogy a keretek kezdetét nem karakterek, hanem egy speciális bitminta jelzi, mégpedig a 01111110. Ha az adatban találunk öt egymás követő egyest, akkor az adatkapcsolati réteg automatikusan egy 0-t szúr be, így ismét egyértelműen meghatározhatóak lesznek a kerethatárok.
Ez már egy jól használható keretezési eljárás, a gyakorlatban azonban mindig a keretszámlálást alkalmazzák valamilyen másik módszerrel együtt. Ezért a vevő egy keretet csak akkor fogad el érvényesnek, ha egyrészt a hibaellenőrző kód alapján helyes, másrészt például a határolójelek is a helyükön vannak.

Nyugtázás

A keretezésnek köszönhetően a vevő értesülhet arról, hogy történt-e átviteli hiba, vagy sem. A forrás azonban semmit sem tud az általa elküldött keretek sorsáról. Ez persze csak akkor jelent gondot, ha összeköttetés alapú szolgálatot kell teljesíteni.
Hogy a kommunikáció biztonságos legyen, kell valamiféle lehetőség arra, hogy a vevő értesíthesse a forrást az általa fogadott adatok állapotáról. Például a forrásnak vissza kell küldeni egy nyugtát, amelyben megerősíti, hogy az adatok rendben megérkeztek, vagy épp azt kéri, hogy ismételje meg az adást.
Vannak bizonyos szerencsétlen körülmények, amelyekkel számolnunk kell. Például mi a helyzet akkor, ha egy keret teljes egészében elveszik? (Ilyen előfordulhat, például amikor nagyon zajos a vonal). A keretek elvesztése rendkívül kellemetlen dolog, mert a forrás várja a nyugtát, a vevő azonban nem is tud arról, hogy neki keretet küldtek, amelyre egy nyugtával kéne válaszolnia. Ezt a problémát persze könnyen kiköszöbülhetjük úgy, hogy a forrás minden kerethez egy időzítőt is rendel. Amikor a keret küldésétől számítva eltelik egy bizonyos idő, és nem érkezik nyugta, akkor valószínűsíthető, hogy a keret nem érkezett meg a célhoz. Ilyenkor a forrás felhagy a várakozással, és megismétli a kérdéses keretet.
Egy keret megismétlése azonban nem veszélytelen dolog, ugyanis fennáll az eshetősége, hogy a vevő kétszer kapja meg ugyanazt a keretet, így kétszer továbbítja azt a hálózati réteg felé. Ezért szükség van keretazonosítók bevezetésére is, például sorszámokra, amelyekkel meg tudjuk számozni az elküldött kereteket.
Az időzítők és a keretek sorszámozásával az adatkapcsolati réteg biztosítani tudja, hogy minden pontosan egyszer (nem kétszer és nem is nullaszor) érkezzen meg a fogadóhoz. A gyakorlatban ez egy kicsit bonyolultabban zajlik a fent leírtaknál. A következő részben részletesen is megnézzük majd, milyen további problémákkal szembesülhetünk a való életben.

Forgalomszabályozás

Egy keretet elveszteni nem csak a zajos átviteli csatorna miatt lehet. Előfordulhat olyan eset is, hogy a küldő sokkal gyorsabban küldi a kereteket, mint ahogy azt a vevő kényelmesen fogadni tudná. Például azért, mert a vevő eleve lassabb, vagy éppen szörnyen leterhelt. Ha ez a helyzet, akkor a forrás teljesen elárasztja a célállomást, amely előbb-utóbb nem lesz képes megfelelően fogadni a kereteket, és egészen biztos, hogy egyet-kettőt el fog veszíteni. Nagyon szomorú, ha ilyesmi megtörténik, ezért szükség van valamiféle védelmi mechanizmusra, amely megakadályozza az ilyen helyzetek kialakulását.
A megoldás kulcsa a vevő visszafogásában rejlik, azaz nem szabad hagyni, hogy az gyorsabban adjon, mint ahogyan azt a célállomás venni képes. Ez az úgynevezett forgalomszabályozás (flow control). Sokféle forgalomszabályozási módszer ismert (a következő részekben részletesen is megnézünk néhányat), de a legtöbb ugyanazt az alapelvet követi: addig nem küldik el a következő keretet, amíg a vevő valamilyen módon engedélyt nem ad rá. Ez nem feltétlenül jelenti azt, hogy a vevőnek minden egyes keret után szólni kell, hogy jöhet a következő. Lehet ugyanis, hogy a kapcsolat felépítésekor megmondja, hogy most például x darab keretet máris lehet küldeni, de utána addig semmit, amíg nem jelez vissza. A keret küldését persze ezen kívül más, az adott protokolltól függő szabályok is befolyásolják. A továbbiakban erről még szólunk részletesebben is.

Hibaészlelés és javítás

A biztonságos kommunikációhoz önmagában nem elég, hogy minden keret pontosan egyszer érkezik meg, illetve hogy a forrás nem küld gyorsabban, mint ahogy a vevő fogadni képes. Legalább ennyire fontos, hogy pontosan az érkezzen meg, amit átküldünk, azaz a bitek értéke ne változzon.
Hogy mikor és mennyi átviteli hiba keletkezik, abban szerepet játszik az is, hogy milyen technikával továbbítjuk a biteket. Például a rádióadók esetében a hibák inkább csoportosan fordulnak elő. Más rendszereknél pedig csak egy-egy bit romlik el, viszont időben egymástól függetlenül.
Mindkét esetnek van jó és rossz oldala is. Ha a hibák csoportosan jönnek, az abból a szempontból jó, hogy nem lesz minden elküldött keretünkben hiba, így mondjuk átlagosan csak minden 100. keretet kell újraküldeni. Hátránya azonban, hogy ha hiba van, akkor az valószínűleg elég sok bitet érint a keretben ahhoz, hogy nehéz legyen kijavítani, esetleg magát a hibát felfedezni.

Akárhogyan is legyen, a fellépő hibákat kezelni, sőt javítani kell. A hibák kezelésére két alapvető módszer létezik. Az első esetben annyi, úgynevezett rendundáns információt fűzünk az adatbitekhez, amennyiből a forrás fel tudja fedezni az esetleges hibát, és ki is tudja következtetni, hogy mi lehetett az eredeti szöveg. Ehhez úgynevezett hibajavító kódok lehetnek a segítségünkre. A másik stratégia szerint csak annyi redundáns bitet fűzzünk az adathoz, amennyiből meg tudjuk állapítani, hogy történt-e hiba, vagy sem. Ehhez hibajelző kódokat kell alkalmaznunk. (Fontos, hogy ebben az esetben csak a hiba meglétére tudunk következtetni, arra nem, hogy melyik bit hibás).

Hogy megérthessük, miként működnek a hibajavító és hibajelző kódok, fontos definiálni azt, hogy „matematikailag" mit is nevezünk átviteli hibának.

A Hamming-távolság

A keretben szereplő adatbitek számát jelöljük m-el, a hozzáfűzött redundánsbitek számát pedig r-el. Az n = r + m hosszú adat- és redundáns bitekből álló bitsorozatot kódszónak nevezzük.

Két kódszót úgy tudunk összehasonlítani egymással, hogy bitenként végigmegyünk rajtuk, és minden azonos helyen lévő bit között elvégzünk egy kizáró vagy (XOR) műveletet. Ha a művelet végeredménye 1, akkor a két bit különböző. Azt, hogy a két kódszó összehasonlításakor hányszor kapunk 1-est, azaz hány eltérő bit van, a két kódszó ***Hamming távolságának*** nevezzük. Ha más szemszögből közelítjük meg a dolgot, akkor akár azt is mondhatnánk, hogy két kódszó Hamming távolsága azt határozza meg, hogy hány darab egybites hiba kell ahhoz, hogy az egyik kódszóból megkapjuk a másikat.

Mivel az adat a bitek bármilyen „kombinációjából" állhat, összesen 2^m-ediken féle úgymond legális adatbit sorozat szerepelhet egy keretben. A redundáns bitek számítási módjából következik, hogy nem fordulhat elő mind a 2^r darab, azaz a lehetséges kódszavak száma nem éri el a 2^n-t. Ha tudjuk a redundáns bitek kiszámításának módját, akkor könnyedén elkészíthetjük a legális kódszavak listáját. Ha megvan a lista, akkor abból kikereshetjük azt a két kódszót, amely a legkisebb Hamming távolsággal rendelkezik. Ez az érték lesz a kód Hamming távolsága.

Hogy egy kód hibajavító, vagy csak hibajelző, azt a Hamming távolsága határozza meg. Ha például fel szeretnénk ismerni d bithibát, akkor legalább egy d + 1 Hamming távolságú kódot kell alkalmaznunk, hiszen egy ilyen kódban hiába fordul elő d bithiba, akkor sem tudunk megkapni egy másik érvényes kódszót.

A hibajavító kódnak azonban nagyobb Hamming távolsággal kell rendelkeznie, pontos értéke (2d + 1), ha ki is szeretnénk javítani legaláb d darab bithibát. Az ekkora távolságú Hamming kódoknál az érvényes kódszavak egymástól olyan messze vannak, hogy d darab bit megváltoztatása sem viheti egy másik legális kódszóba.

Ezt csak elmesélni bonyolult, valójában nagyon egyszerű dologról van szó. Nézzünk meg egy-egy példát a hibajelző, illetve a hibajavító kódra. A legegyszerűbb hibajelző kód a paritásbit, amely azt mondja meg, hogy az adattagban lévő 1-esek száma páros, vagy páratlan. (Ebben az esetben a redundáns információ maga a paritásbit). Ha egy bithiba történik, akkor megváltozik a paritás, így azt ezzel a módszerrel észlelni tudjuk. A paritás bites kód Hamming távolsága 2, mivel egy bit hibát észlelhetünk. (Például két bithibára ezzel a módszerrel nem derülne fény).

Nézzünk egy példát a hibajavító kódra is. Példánk alanya egy olyan kód, amely a következő legális kódszavakat fogadja el: 0000000000, 0000011111, 1111100000, 1111111111. Ennek a kódnak a Hamming távolsága minden bizonnyal 5, mivel kétbitnyi hibát tud kijavítani. Ha például a 0000000111 kódszó érkezik, akkor ebből ki tudjuk következtetni, hogy az eredeti üzenet a 0000011111. Ha azonban három vagy több bithiba keletkezik, például a csupa nulla helyett szintén a 0000000111-t kapjuk, akkor nem a helyes kódszót állítjuk vissza.

A gyakorlatban is használt hibajavító kódok közül a Hamming kódot emelnénk ki. Működésének ismertetése túlmutat e sorozat keretein, annyit jegyeznénk meg róla csupán, hogy egybites hibák kijavítására szolgál, de némi trükközés segítségével képessé tehetjük csoportos hibák javítására is. A hibajavító kódokkal a való életben ritkán találkozhatunk. Ennek az az oka, hogy sokkal gazdaságtalanabb, mintha csak hibajelző kódot használnánk, és ha szükséges, újra küldenénk a hibás keretet. A hibajavító kódoknak egyirányú kommunikáció esetén van jelentősége, akkor ugyanis a vevő nem tud visszaszólni a forrásnak.

Hibajelző kódok

A hibajelző kódok közül már volt szó a paritásbites kódról, amelynek nagy hátránya, hogy csak páratlan számú hibáknál bukik ki a probléma, azaz egy hiba jelzésének az esélye 50%. Ez nem egy jó arány. Sokat segíthet a dolgon, ha az elküldendő blokkot egy n X m -es mátrixnak tekintjük, és minden oszlophoz külön kiszámoljuk a paritásbitet. Ezután a mátrixhoz csatoljuk utolsó sorként a paritásbiteket, majd sorfolytonosan továbbítjuk az egész mátrixot.

Ez már egy némileg elfogadható megoldás, a gyakorlatban mégis inkább a polinomkódot, vagy ismertebb nevén a CRC kódot választják. A CRC kód működésének ismertetése is túlmutat e sorozat keretein, csupán annyit jegyeznénk meg róla, hogy nagyon pici a lehetősége a hibák rejtvemaradásának. A CRC16 például 16 bites ellenőrző összeggel dolgozik, így felismer minden egy- és kétbites hibát, továbbá minden páratlan számú bitet érintő hibát és minden 16, vagy annál kevesebb bitnyi csoportos hibát. (A csoportos hiba alatt azt értjük, hogy nem feltétlenül rossz az összes bit, de az első és az utolsó biztosan az).

A CRC-16 a 17 bites csoportos hibák 99,997%-át is felismeri. A CRC kicsit bonyolult algoritmus (főleg, hogy sok polinomosztást kell végrehajtanunk), mégis könnyű készíteni olyan áramkört, amely képes kiszámolni az ellenőrző összeget. Ezért általában hardveresen végzi a CRC-n alapuló ellenőrzést.

A következő részben tovább folytatjuk az adatkapcsolati réteggel való ismerkedést, méghozzá elemi adatkapcsolati protokollokat veszünk majd szemügyre.

Garzó András
garzo@interware.hu