Számítógéphálózatok – 7. előadás

TCP - Transmission Control Protocol

- Az alapja: **RFC793** - Postel
- A leggyakrabban használt 4. szintű protokoll
- Kapcsolat-orientált: mindig pontosan két partner közti kommunikáció
  - Következmény: multicast-on nincs értelme
  - Analógia: telefon
- A megbízhatatlan IP fölött megbízható szolgáltatást nyújt
- Alkalmazás-alkalmazás kommunikációt tesz lehetővé (szemben az IP host-host kommunikációval)
- Manapság is fejlesztik

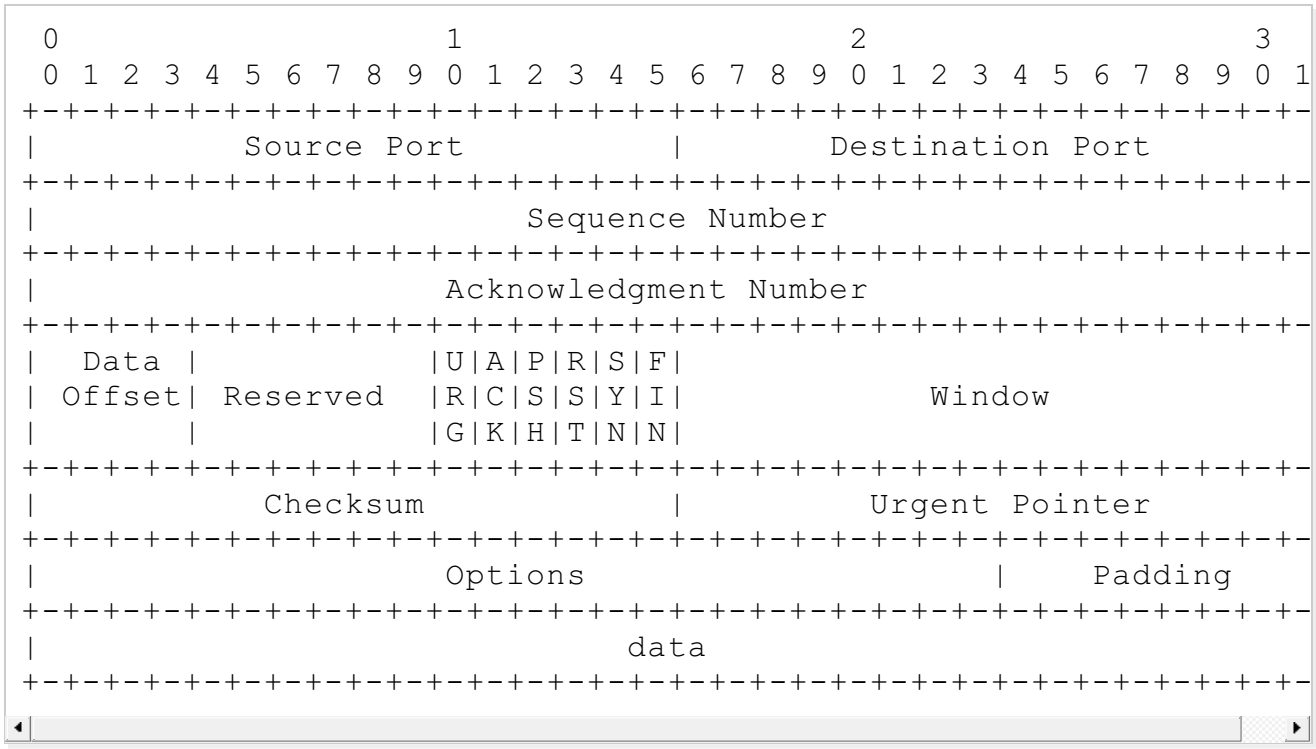
TCP szolgáltatások

- A TCP darabokra bontja az információt. Egy IP rétegnek átadott darab: szegmens
- Pozitív nyugtázás
- Minden szegmensre nyugtát vár
- Ha nem jött nyugta, újraküld
  - Nem feltétlenül minden szegmenst külön-külön
  - Nem feltétlenül azonnal
- Minden szegmenst csekkszumma véd. Ha ez hibát jelez, egyszerűen eldobja
  - Ilyenkor a küldő oldal timeout-ol és újra küld
- Az IP csomagok nem feltétlenül sorrendben érkeznek meg. A TCP helyreállítja a sorrendet
- IP csomagok duplázódhatnak. A TCP kiszűri a duplikátumokat
- A TCP folyamvezérlést - flow controlt - alkalmaz: az adatáramlás sebességét fékezni és gyorsítani tudja a rendelkezésre álló erőforrásoktól függően
- A TCP byte (oktet) folyamot visz át: nincs rekord vagy sorhatár fogalom. Ilyesmiről a felsőbb szintnek kell gondoskodni, ha szükséges
  - Egy byte folyamot öntünk be az egyik oldalon, a TCP gondoskodik róla, hogy pont az a byte folyam bukik ki a másik oldalon
- A TCP full duplex kapcsolatot tesz lehetővé: mindkét irányban és egymástól függetlenül áramlik a byte folyam

Portok

- Multiplexálás/demultiplexálás
- 16 bites port szám
- 0-1023 well known ports
- 1024-től szabadon használható

TCP fejrész



- A kapcsolatot ez a négyes határozza meg: *source és destination port, source és destination IP cím* (telefon hívás analógia)
  - Az egyik oldalon az IP cím/port páros: socket
  - A kapcsolatot egy socket pár határozza meg
- Sequence number: a byte folyamban az ebben a szegmensben küldött első byte sorszáma
  - Körbefordul: modulo 2^{32} mutatja a sorszámot
  - A kapcsolat elején rendszerint **nem** 0
    - ISN, initial sequence number: a kezdeti érték
  - Hagyományosan nem is véletlen, megjósolható
    - Támadásra ad módot, manapság véletlen szám
      - Tűzfalak (pl. OpenBSD) külön erőfeszítéseket tehetnek a „véletlenség” növelésére
  - SYN, synchronize flag: a kapcsolat felépítésekor használatos
    - Egy sequence numbert „elfogyaszt”
- Acknowledgment number: a vett byte-folyamot eddig a sorszámig nyugtázom. A következő venni kívánt sorszámot tartalmazza
  - Az acknowledgement number akkor érvényes, ha az ACK bit áll
  - Az ACK bit rendszerint áll: ha nem jött új adat (vagy jött, de nem akarom nyugtázni), megismétlem az előző acknowledgement numbert
  - A TCP sliding-window (csúszó ablakos) nyugtázást alkalmaz, szelektív és negatív nyugták nélkül
    - Ha kimaradt egy csomag, de a következő már megjött, nincs mód arra, hogy a megérkezettet nyugtázzam
    - Ha megjött egy csomag, mondjuk 2001-től 3000-ig de hibás, nincs erre más mondásom, mint az, hogy újra nyugtázom 2000-ig a folyamot
- Data offset: a fejrész hosszát adja meg 4 byte-os egységekben.
  - Az opciók miatt változhat a fejrész hossza
  - Rendszerint 20, legfeljebb 60
- Flagek
  - URG: az urgent (sürgős adat) pointer érvényes
  - ACK: az acknoledgement number érvényes
  - PSH: azonnal add fel a felső rétegnek a szegmenst (push)
  - RST: lecsaptam a kagylót, durva bontás (reset)

    - SYN: szinkronizáljuk a sequence numbereinket, kapcsolat indítása
    - FIN: befejeztem az adatküldést, leteszem a kagylót (finish)
        - Adatot fogadni még tudok
- Window: ablak. Az acknowledgement numbertől ennyi byte fogadására készen állok
- Csekkszumma: a szokásos one's complement 16 bites darabokra, a teljes szegmensre
    - Nem csak az TCP csomagot, hanem egy „pszeudó fejrészt” is figyelembe vesz:

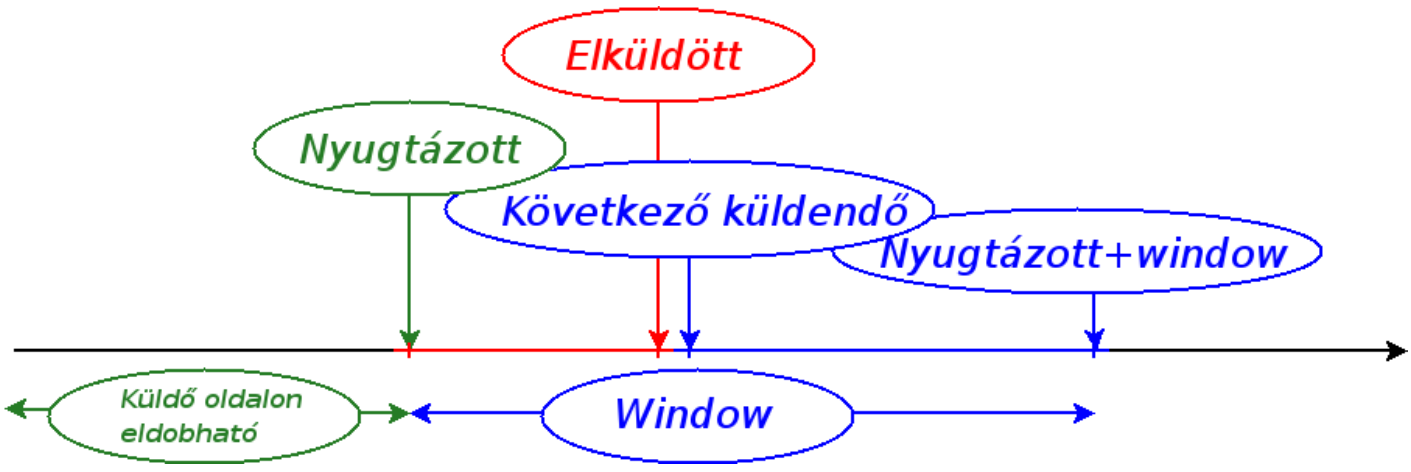
```
+--------+--------+--------+--------+
|           source address           |
+--------+--------+--------+--------+
|         destination address        |
+--------+--------+--------+--------+
|  zero  |protocol|   TCP length    |
+--------+--------+--------+--------+
```

- Az IP fejrészből ismétel meg elemeket
    - A rossz helyre küldött TCP csomagokat lehet így kiszűrni
    - UDP-nél is van ilyen
- Urgent pointer: ha áll az URG bit, akkor az adatban eddig a pontig sürgős adat van
- Pl. fájl transzfer abortálásra szolgál
- Opciók
    - típus: 1 byte-os opció
    - típus, hossz, opció adatok
    - Gyakran használt opció: MSS, Maximum Segment Size
- Adat: nincs feltétlenül
    - Például ha csak nyugtázunk vett adatot, akkor nincs.

TCP opciók: Type/Length/Value

```
                        +-----+
End of option list:     |  0  |
                        +-----+
                        1 byte

                        +-----+
No operation:           |  1  |
                        +-----+
                        1 byte

                        +-----+-----+--------+
Maximum segment size:   |  2  |  4  | MSS    |
                        +-----+-----+--------+
                        4 byte

                        +-----+-----+-----+
Window scale factor:    |  3  |  3  |  x  |
                        +-----+-----+-----+
                        3 byte
                        0 <= x <= 14 : shift count
                        x bittel elshifteljük a 16 bites window-t

                        +-----+-----+------------+-------------+
Timestemp:              |  4  | 10  | Time value | Time echo   |
                        +-----+-----+------------+-------------+
                        10 byte
                        Az ACK-val együtt visszaküldjük a kapott é
                        RTT számolást pontosít
```

Sliding window



- A nyugtázott+window sorszámig folyamatosan küldhető adat
  - Rendszerint nem tölti ki az egész window-t
- A nyugtázott+window sorszámnál nagyobb sorszámú oktet nem küldhető
- A window folyamatosan csúszik jobbra az ábrán
- A window-t a fogadó csökkentheti/növelheti
  - Ha nyugtáz és nem akarja növelni, akkor a nyugtában kisebb window-t kell mondania az előzőnél

- A window mérettel a fogadó szabályozhatja a küldés ütemét
  - Flow control: az alkalmazás igényeihez igazodhat
    - Pl. a window lehet az alkalmazás által vételre felkínált buffer mérete
  - Congestion control: torlódás és csomagvesztés elkerülésére
- Egy nyugta mindig addig az SN-ig nyugtáz mindent, nem csak az utolsó csomagot
- A window nyílik, ha a jobb széle jobbra mozdul
- A window becsukódik:
  - ha a hirdetett window 0,
  - az adó kimerítette a küldhető adatmennyiséget,
  - a vevő mindent nyugtázott
- A window zsugorodik (shrinks), ha a jobb széle balra mozdul
  - Lehetséges, de ellenjavalt
  - A TCP-ben ez különleges dolog: vannak protokollok, ahol ez lehetetlen
- Újraküldés
  - Ha három ugyanolyan ACK-t kapok, onnan kezdve újraküldök
  - Ha timeout (RTO) lejár, és nem kapok ACK-t

TCP kapcsolatfelépítés

- Three-way handshake
  1. A kezdeményező (kliens) küld egy SYN flages csomagot. Eldől a kapcsolatra
     - A hívó fél ISN-je
     - portok. A hívó port többnyire véletlenszerűen választott
  2. A hívott (szerver) küld egy ACK-t, és SYN-t tartalmazó csomagot. Nyugtázza a kapott csomagot, a SYN elfogyaszt egy sequence numbert. Eldől:
     - A másik ISN
  3. A hívó küld egy ACK-t, nyugtázza a másik csomagját. Az a SYN is elfogyaszt egy sequence numbert.
- A kezdeményezőre azt mondjuk: active open-t hajt végre
- A hívott: passive open-t
- Ha a kapcsolat felépítés nem sikerült, a kezdeményező timeout után újra próbálkozik
- Ha többedszerre sem sikerül, értesíti az alkalmazást a kudarcról

TCP SYN flood támadás

- A támadott IP címre SYN csomagok tömegét küldik
  - A forrás IP címek változnak, hamisak, véletlenszerűen generáltak
  - A támadott állomás SYN/ACK-t küld és vár az ACK-ra, ami sose jön meg
  - Elemészti a támadott állomás erőforrásait
  - Megbéníthat egy állomást (DoS)
- Védekezés
  - Egy hálózatból kifele csak az oda tartozó source címekkel mehessenek ki csomagok
    - **BCP38**, **BCP46**
  - Syn cookie
    - A SYN/ACK-val küldött sequence number ravaszul választott, és küldésekor nem foglalunk semmi erőforrást
      - ISN = *f(source addr, source port, dst addr, dst port, time, secret)*
      - Lásd: **http://cr.yp.to/syncookies/archive**
    - A visszaküldött ACK tartalmazza ezt a cookie-t, és csak ennek érkezésekor foglalunk erőforrást

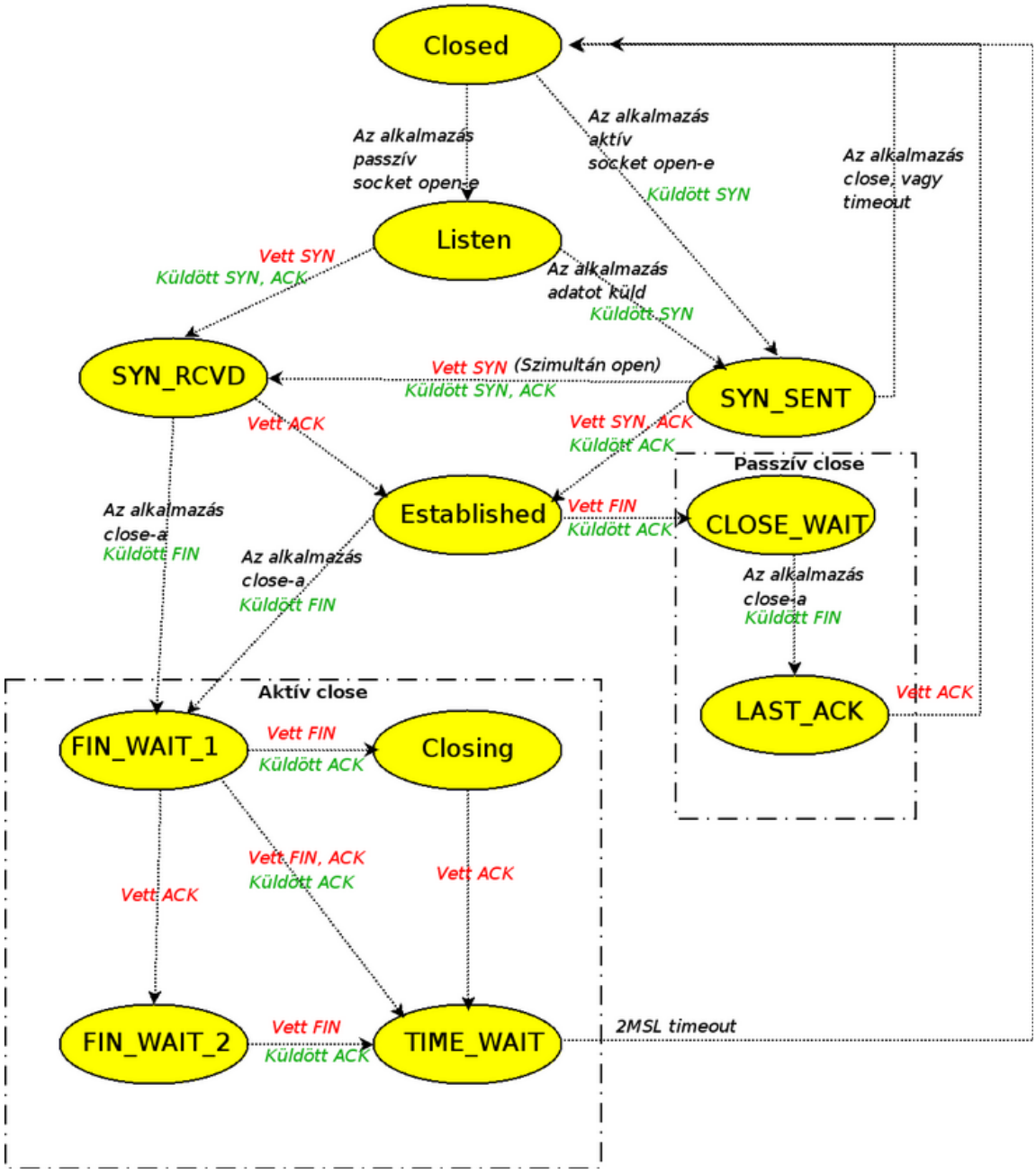
Kapcsolat zárás

- A TCP kapcsolat full duplex, ezért a FIN csak azt jelenti: én nem küldök több adatot
- TCP half close: csak az egyik irányban zárt kapcsolat
  - Elvben lehetséges, gyakorlatban ritka
- Négy csomag utazhat
  1. Az egyik *(A)* fél küld egy FIN-t
     - A másik *(B)* értesíti az applikációt, hogy vége az adatnak (EOF, End of File)
  2. *B* nyugtázza ACK-val
  3. *B* is küld FIN-t, ha befejezte az adatküldést
  4. Erre is megjön a nyugta
- A FIN-is elfogyaszt egy sequence numbert
- A activ close-t, B passive close-t hajt végre

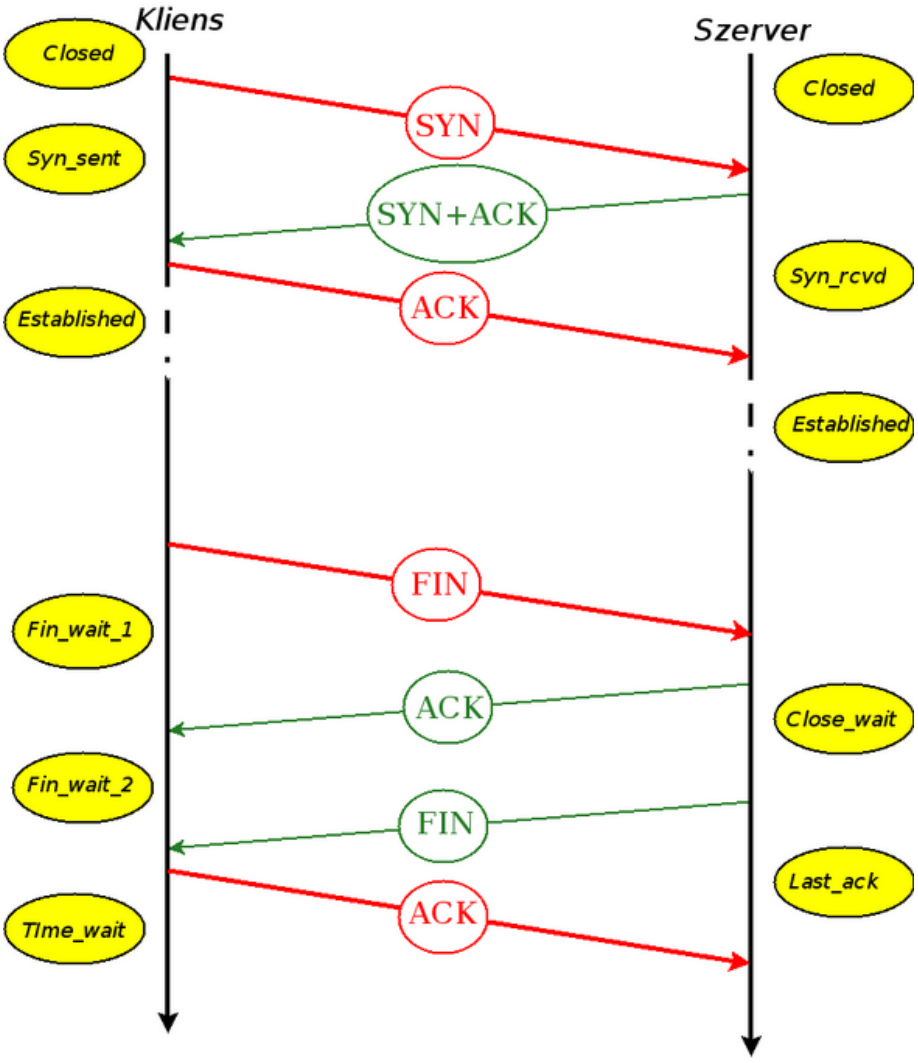
MSS opció

- A SYN-nel szokás küldeni
- Ez a legnagyobb szegmens, amit venni szeretnék
- Nincs garancia rá, hogy útközben nincs kisebb MTU
- Path MTU discovery választ adhat a valódi értékre

TCP állapotábra



A szokásos folyamat



TIME_WAIT állapot

- Az aktív close-t végző utolsó állapota
- Lehet, hogy elvész az utolsó küldött ACK
- Ilyenkor a passzív oldal újra küldi a FIN-t
  - Ezt egy másik kapcsolat részeként lehet felfogni, RESET lenne a válasz
- MSL: Maximum Segment Lifetime. Becsült érték: ennél tovább nem lehet a hálózaton egy TCP szegmens
- TIME_WAIT állapot = 2MSL állapot: 2xMSL ideig van ebben egy kapcsolat, utána ,,closed"
- Lehet, hogy amiatt nem indul el egy szerver, mert nem tudja megnyitni ,,listen"-re a socket-et: TIME_WAIT-ben van
  - Implementáció függő
- Nem csak a TIME_WAIT, hanem a FIN_WAIT_2 is olyan, hogy benneragadhatunk: timeout után fel kell szabadítani az erőforrásokat, alapállapotba kell menni

Quiet time elv

- Ha egy gép újraindul, akor a TIME_WAIT-ben levő kapcsolatairól nem tudhat
- Ezért 2xMSL ideig újrainduláskor ne létesítsen TCP kapcsolatot
- A gyakorlatban a boot idő nagyobb

Reset

- Akkor küldik, ha olyan csomag érkezik, amit nem találunk szabályosnak
- Például ha olyan portra érkezik TCP kérés, amin nem figyel alkalmazás
- Kapcsolat erőszakos bontására is használható
  - Ilyenkor nincs garancia arra, hogy minden csomagot megkapott az alkalmazás: abort

Half open kapcsolatok

- Ha TCP kapcsolatban álló állomások közül az egyik (vagy rajta az alkalmazás) újraindul, a másik fél élőnek hiheti a kapcsolatot
- Küldhet adatot, ez meglepetés lesz az újra indult oldalon
- Ilyenkor RESET a válasz

Szimultán open

- Keresztbe küldhet két alkalmazás SYN-t ugyanarra a socket párra
- Ilyenkor mindkettő ACK-val válaszol
- A kapcsolat felépül és csak egy kapcsolat épül fel!

Szimultán close

- A küldött FIN-re nem ACK, hanem FIN jön
- ACK-val meg kell válaszolni, és várni az ACK-ra: CLOSING állapot
- Ilyenkor mindkét oldal TIME_WAIT állapotba kerül

TCP reset támadás (TCP Reset attack)

- *A* és *B* közt élő TCP kapcsolatba *E* RESET csomagot csempészhet
- *E*-nek tudnia kell a következőket:
  - source/destination IP cím
  - source/destination port
  - sequence number
- 2004 májusában nagy port vert fel
- Paul Watson - Slipping in the Window: TCP Reset Attacks , CanSecWest konferencián
- Felfedezte, hogy nem szükséges a kurrens sequence numbert tudni: elég a window-ban levő bármi!
  - Nagy window méretnél (ami egyre gyakoribb) könnyebb a támadás
  - Brute force támadás is indítható
- BGP kapcsolatok nagyon veszélyeztetettek

TCP szerverek

- Az egyes ismert szolgáltatásokhoz a IANA portokat rendel: *well-known ports*, pl:
  - 20 - FTP data
  - 21 - FTP control
  - 23 - TELNET
  - 25 - SMTP
  - 53 - DNS
  - 80 - HTTP
- Az implementáció, sőt a konfiguráció mást is választhat
- Unixokban az `/etc/services` tartalmazza az ismert portok nevét és számát
- A szerverek induláskor figyelnek a választott porton (Listen állapot)

    - Ha több IP cím van, általában mindegyiken
    - Lehet, hogy csak az IP címek egy részén
    - Lehet, hogy ugyanazon gépen különböző IP címeken különböző programok figyelnek
- Ha kérés érkezik, a szerver egy gyerek processzt forkol
    - Azé lesz az új kapcsolat
- A netstat parancs mutatja a pillanatnyi kapcsolatokat
    - Alternatíva: `lsof -i`

`inet` daemon

- Unixokon klasszikus, egyszerű szolgáltatások indításának eszköze
- Belső szolgáltatások (pl. echo) és tetszőleges program indítása
    - Az indított program standard inputja és outputja a TCP kapcsolat lesz
- Alternatíva: `xinetd`
    - Árnyalni lehet, hogy milyen feltételekkel, honnan fogad el kapcsolatot
        - Hívó IP cím
        - Idő: mettől meddig
    - Árnyalni lehet az induló processz környezetét

TCP daemon, `tcpd`, alias `tcp_wrapper`

- `inetd` és egyes programok közé ékelődhet
- Árnyaltan lehet akciót kezdeményezni
    - Elutasítani/elfogadni a kapcsolatot
    - Naplózni
    - Figyelmeztető levelet küldeni, tetszőleges parancsot kiadni
- Két fő konfigurációs fájl: `/etc/hosts.allow, /etc/hosts.deny`
    1. Az első match nyer
    2. Ha nincs match, engedi a kapcsolatot
    3. A fájlokat a következő sorrendben nézi: `hosts.allow, hosts.deny`
- Nem csak inetd mögött, hanem libwrap-pal egybelinkelt bármilyen programmal használható
    - Az ssh szerver ilyen szokott lenni

Socket server példa program

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <time.h>

int main(int argc, char *argv[])
{
    int listenfd = 0, connfd = 0;
    struct sockaddr_in serv_addr;

    char sendBuff[1025];
    time_t ticks;

    listenfd = socket(AF_INET, SOCK_STREAM, 0);
    memset(&serv_addr, '0', sizeof(serv_addr));
    memset(sendBuff, '0', sizeof(sendBuff));

    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
    serv_addr.sin_port = htons(5000);

    bind(listenfd, (struct sockaddr*)&serv_addr, sizeof(serv

    listen(listenfd, 10);

    while(1)
    {
        connfd = accept(listenfd, (struct sockaddr*)NULL, NU

        ticks = time(NULL);
        snprintf(sendBuff, sizeof(sendBuff), "%.24s\r\n", ct
        write(connfd, sendBuff, strlen(sendBuff));

        close(connfd);
        sleep(1);
     }
}
```

Socket client példa program

```
#include <sys/socket.h>
#include <sys/types.h>
```

```
#include <netinet/in.h>
#include <netdb.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <arpa/inet.h>

int main(int argc, char *argv[])
{
    int sockfd = 0, n = 0;
    char recvBuff[1024];
    struct sockaddr_in serv_addr;

    if(argc != 2)
    {
        printf("\n Usage: %s  \n",argv[0]);
        return 1;
    }

    memset(recvBuff, '0',sizeof(recvBuff));
    if((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    {
        printf("\n Error : Could not create socket \n");
        return 1;
    }

    memset(&serv_addr, '0', sizeof(serv_addr));

    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(5000);

    if(inet_pton(AF_INET, argv[1], &serv_addr.sin_addr)<=0)
    {
        printf("\n inet_pton error occured\n");
        return 1;
    }

    if( connect(sockfd, (struct sockaddr *)&serv_addr, size
    {
       printf("\n Error : Connect Failed \n");
       return 1;
    }

    while ( (n = read(sockfd, recvBuff, sizeof(recvBuff)-1)
    {
        recvBuff[n] = 0;
        if(fputs(recvBuff, stdout) == EOF)
        {
            printf("\n Error : Fputs error\n");
        }
    }
```

```
    if(n < 0)
    {
        printf("\n Read error \n");
    }

    return 0;
}
```

Szerző: Pásztor Miklós, Máray Tamás