Laboratorio di Programmazione in Rete

a.a. 2005/2006

http://www.di.uniba.it/~lisi/courses/prog-rete/prog-rete0506.htm

dott.ssa Francesca A. Lisi
lisi@di.uniba.it

Orario di ricevimento: mercoledì ore 10-12

N.B. Il presente materiale didattico è stato
prodotto da: dott.ssa V. **Carofiglio**
rielaborato da: dott.ssa F.A. **Lisi**

dib

Opzioni per le Socket

- Ogni socket aperta ha delle proprietà che ne determinano alcuni comportamenti
- Le opzioni della socket consentono di modificare tali proprietà

- Ogni opzione ha un valore di default:
  - Alcune opzioni sono binarie (on/off)
  - Altre hanno un valore (int o anche strutture più complesse)

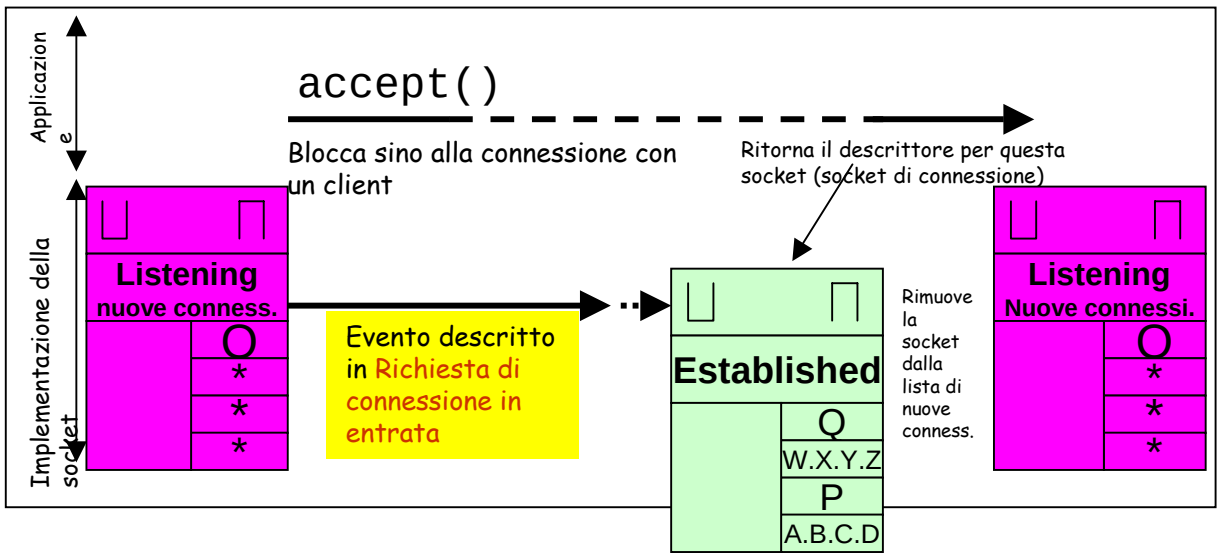
- Le opzioni delle socket sono controllate mediante primitive:
  - → **`getsockopt()`** e **`setsockopt()`**, per configurare alcune caratteristiche proprie solo delle socket
  - → **`ioctlsocket()/fcntl()`**, per configurare caratteristiche comuni a tutti i descrittori di I/O, quali comportamento bloccante o non bloccante.

Vedi Par. 5.1 del libro di Donahoo & Calvert

Opzioni per le Socket: Impostazione delle opzioni generiche

Le opzioni tipiche per le socket possono essere impostate solo quando la socket è resa disponibile all'interfaccia di programmazione

Es: socket di connessione ottenuta da un server in risposta ad una chiamata alla `accept()`



la socket creata verrà resa disponibile al programmatore solo quando questo farà eseguire la `accept()`.

Per rendere possibile configurare la socket anche in questa situazione temporanea, l'interfaccia socket implementa la politica seguente:
la socket di connessione eredita dalla socket di ascolto alcune opzioni, invece di assumere le opzioni di default.

Opzioni per le Socket:

Le funzioni setsockopt() e getsockopt()

Consentono rispettivamente di impostare o rilevare le opzioni di una socket valida

```
int setsockopt( SOCKET s,
                int level,
                int optname,
                const char* optval,
                int optlen );
```

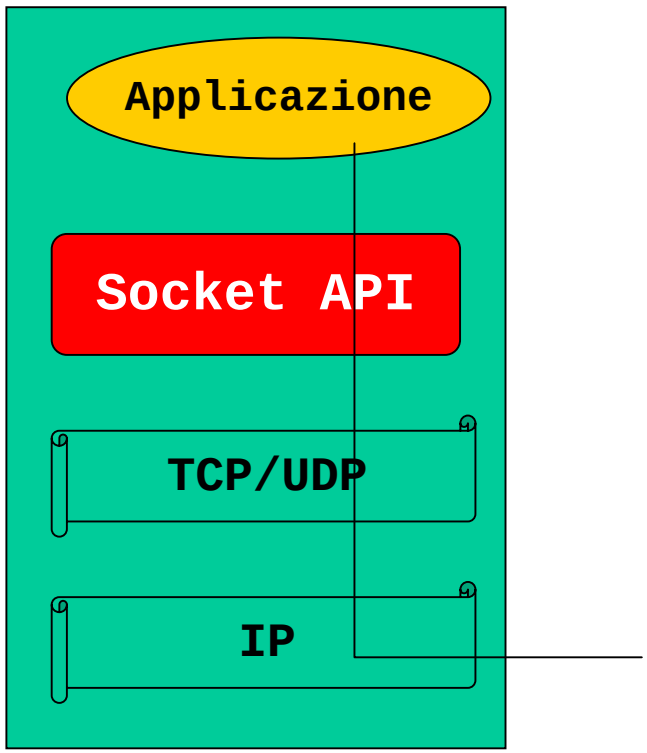
- descrittore di socket
- Livello a cui è definita l'opzione
- Nome dell'opzione
- Pt. al buffer con il valore da assegnare all'opzione
- Dimensione del buffer

```
int getsockopt( SOCKET s,
                int level,
                int optname,
                const char* optval,
                int optlen );
```

- descrittore di socket
- Livello a cui è definita l'opzione
- Nome dell'opzione
- Pt. al buffer con il valore da assegnare all'opzione
- Dimensione del buffer

Restituiscono 0 in caso di successo. Il codice di errore altrimenti

Opzioni per le Socket: Livello delle Opzioni



strutturazione a livelli dei protocolli di rete
l'interfaccia delle socket può usarne più di uno.

1. **SOL_SOCKET** livello socket
2. **IPPROTO_IP** livello IP
3. **IPPROTO_IPV6** livello IPv6
4. **IPPROTO_ICMPV6** livello ICMP (messaggi di controllo)
5. **IPPROTO_TCP** livello TCP

Si avranno allora funzionalità e caratteristiche diverse per ciascun protocollo usato da una socket,

quindi saranno anche diverse le opzioni che si potranno impostare per ciascuna socket, a seconda del *livello* (trasporto, rete, ecc.) su cui si vuole andare ad operare.

Opzioni per le Socket: Opzioni del livello SOL_SOCKET

Value	Type	Meaning
SO_BROADCAST	BOOL	Enables transmission and receipt of broadcast messages on the socket.
SO_CONDITIONAL_ACCEPT	BOOL	Enables sockets to delay the acknowledgment of a connection until after the **WSAAccept** condition function is called.
SO_DEBUG	BOOL	Records debugging information.
SO_DONTLINGER	BOOL	Does not block close waiting for unsent data to be sent. Setting this option is equivalent to setting SO_LINGER with **l_onoff** set to zero.
SO_DONTROUTE	BOOL	Does not route: sends directly to interface. Succeeds but is ignored for both AF_INET and AF_INET6 sockets.
SO_GROUP_PRIORITY	int	Reserved.
SO_KEEPALIVE	BOOL	Sends keep-alives. Not supported on ATM sockets (results in an error).
SO_LINGER	**LINGER**	Lingers on close if unsent data is present.
SO_OOBINLINE	BOOL	Receives OOB data in the normal data stream. (See section Protocol Independent Out-Of-band Data for a discussion of this topic.)
SO_RCVBUF	int	Specifies the total per-socket buffer space reserved for receives. This is unrelated to SO_MAX_MSG_SIZE or the size of a TCP window.
SO_REUSEADDR	BOOL	Allows the socket to be bound to an address that is already in use. (See **bind**.) Not applicable on ATM sockets.
SO_EXCLUSIVEADDRUSE	BOOL	Enables a socket to be bound for exclusive access. Does not require administrative privilege.
SO_SNDBUF	int	Specifies the total per-socket buffer space reserved for sends. This is unrelated to SO_MAX_MSG_SIZE or the size of a TCP window.
SO_UPDATE_ACCEPT_CONTEXT	int	Updates the accepting socket with the context of the listening socket.
PVD_CONFIG	Service Provider Dependent	This object stores the configuration information for the service provider associated with socket *s*. The exact format of this data structure is service provider specific.

Dichiarate in winsock2.h

Opzioni per le Socket: Opzioni del livello SOL_SOCKET (cont.)

SO_BROADCAST: abilita o disabilita la possibilità per una socket di spedire messaggi broadcast. Viene applicato solo alle socket datagram (DGRAM) e solo se la rete sottostante lo permette. **Per default questa opzione è disabilitata**, per impedire ad un processo di spedire accidentalmente un datagram in broadcast.

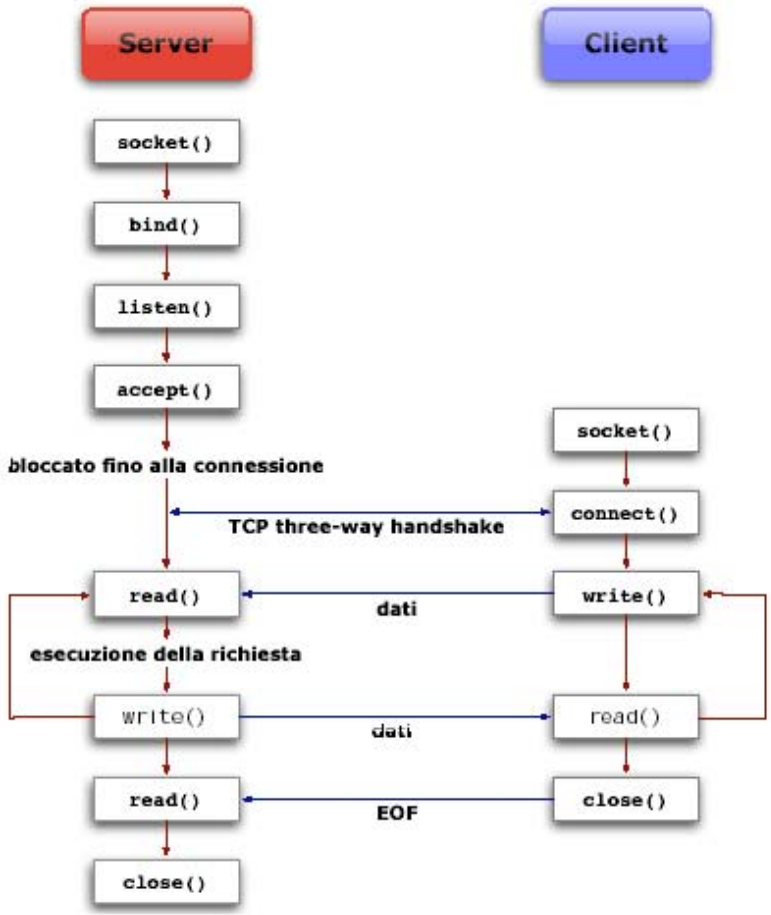
SO_DEBUG è supportata solo da TCP, e ordina al kernel di mantenere informazioni su tutti i pacchetti spediti o ricevuti da/a un certo socket, in una coda circolare.

SO_KEEPALIVE: è applicata solo agli stream TCP, per verificare se una connessione che da molto tempo non scambia dati debba essere chiusa o no. Il motivo per cui una connessione deve essere chiusa da un terminale e' che l'altro terminale o a) e' fuori uso, o b) non e' raggiungibile (rete partizionata o problema nel routing), o c) non e' piu' interessato a quella connessione.

Opzioni per le Socket: Opzioni del livello SOL_SOCKET (cont.)

SO_RCVBUF e SO_SNDBUF: servono a modificare la dimensione dei buffer di ricezione e trasmissione del TCP e dell'UDP.
Per il TCP la dimensione del buffer di ricezione viene inviata durante l'instaurazione della connessione.
E' quindi necessario che per il server questa opzione sia settata prima della chiamata alla `listen()`, mentre per il client deve essere settata prima della chiamata alla `connect()`.

Invece per UDP il buffer di ricezione determina la dimensione massima dei datagram che possono essere accettati.



TCP

Opzioni per le Socket: Opzioni del livello SOL_SOCKET (cont.)

SO_REUSEADDR e SO_REUSEPORT: permettono di effettuare la `bind()` su porte e indirizzi IP gia' utilizzati da qualche altro processo.

La SO_REUSEADDR ad es. puo' essere utile nei seguenti casi:

- si cerca di fare la `bind()` per una socket di ascolto che e' stata chiusa e si vuol fare ripartire, quando ancora esiste una socket di connessione derivata dalla socket di ascolto appena chiusa.

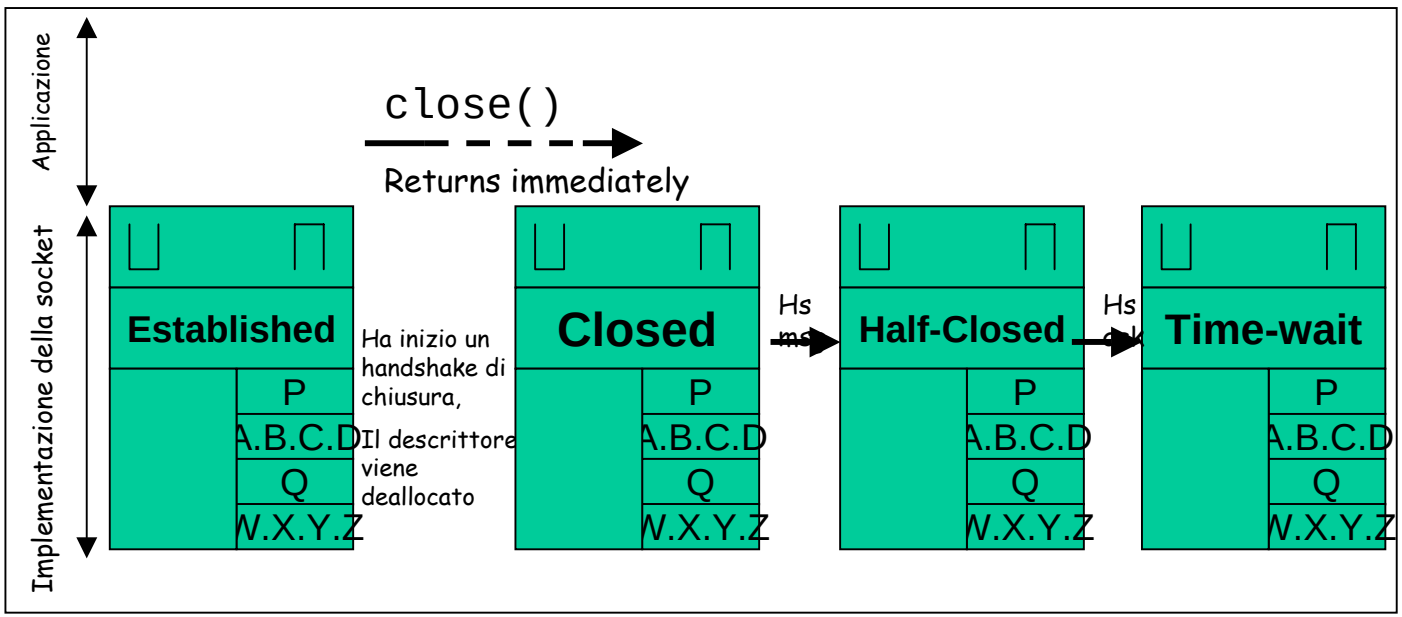
- ci sono più socket di connessione che lavorano sulla stessa porta di un host ma con IP diversi. (es: web server che devono lavorare sulla stessa well know port 80 ma su interfacce diverse)

SO_TYPE: usata solo in lettura, restituisce il tipo della socket, SOCK_STREAM o SOCK_DGRAM.

Opzioni per le Socket: Opzioni del livello SOL_SOCKET (cont.)

SO_LINGER: Determina le modalita' di chiusura realizzate dalla funzione `close()` (`closesocket()` per winsock).

Per default la `close()` restituisce subito il controllo al chiamante, ma se alcuni dati rimangono nei buffer di spedizione il TCP tenta di spedirli.



Opzioni per le Socket: Opzioni del livello SOL_SOCKET (cont.)

Come argomento **optval** in `setsockopt()` viene usato un puntatore ad una struttura di tipo:

```
typedef struct linger {
   int l_onoff; /* 0=OFF, nonzero= ON*/
   int l_linger; /*timeout*/ } linger;
```

se l_onoff==0 l'opzione SO_LINGER e' disabilitata: chiusura di default

se l_onoff != 0 e **l_linger==0** quando una socket chiama la `close()`, il TCP chiude la connessione in modo traumatico, non spedendo i dati eventualmente bufferizzati per la spedizione. Non si va nello stato TIME_WAIT e si rischia di danneggiare l'apertura di una nuova connessione con gli stessi indirizzi IP e di porta.

In entrambi i casi non sappiamo se il TCP peer ha ricevuto tutti i dati, e nemmeno se li ha ricevuti l'application peer.

Opzioni per le Socket: Opzioni del livello SOL_SOCKET (cont.)

```
typedef struct linger {
   int l_onoff; /* 0=OFF, nonzero= ON*/
   int l_linger; /*timeout*/ } linger;
```

se **l_onoff** != **0** e **l_linger** !=**0** quando una socket chiama la `close()`, se la socket e' di tipo bloccante (e' il default) il TCP tenta di spedire i dati eventualmente bufferizzati per la spedizione, fino a che si verifica una di queste condizioni:

- tutti i dati sono trasmessi e riscontrati dal TCP peer, ed allora la funzione `close()` restituisce il controllo al chiamante con risultato 0, passando dallo stato TIME_WAIT.

OPPURE

- scade il tempo assegnato di attesa `l_linger` e la funzione `close()` restituisce -1 mandando un segment RST (reset) all'altro terminale, e non passa dallo stato TIME_WAIT.

Opzioni del livello SOL_SOCKET: Esempio 1

```
int rcvBufferSize;
int sockOptSize;
.
.
/*preleva la dimensione di origine del buffer*/
sockOptSize = sizeof(rcvBuffersize);
If(getsockopt(m_socket, SOL_SOCKET, SO_RCVBUF,
               &rcvBufferSize, &sockOptSize) < 0)
       /*gestione errore*/
printf("initial receive buffer size: %d\n", rcvBufferSize);

/*raddoppia la dimensione del buffer*/
rcvBufferSize *=2;
If(setsockopt(m_socket, SOL_SOCKET, SO_RCVBUF,
               &rcvBufferSize, sizeof(rcvBufferSize)) < 0)
       /*gestione errore*/
printf("current receive buffer size: %d\n", rcvBufferSize);
```

Opzioni del livello SOL_SOCKET: Esempio 2

```
#include <stdio.h>
#include "winsock2.h"
void main()
{ //----------------------------------------
  // Declare variables
  WSADATA wsaData;
  SOCKET ListenSocket;
  sockaddr_in service;

//----------------------------------------
// Initialize Winsock
int iResult = WSAStartup( MAKEWORD(2,2), &wsaData );
if( iResult != NO_ERROR )
printf("Error at WSAStartup\n");

//----------------------------------------
// Create a listening socket
ListenSocket = socket( AF_INET, SOCK_STREAM, IPPROTO_TCP );
if (ListenSocket == INVALID_SOCKET)
{ printf("Error at socket()\n");
  WSACleanup();
  return; }
```

Opzioni del livello SOL_SOCKET: Esempio 2 (cont.)

```
//----------------------------------------
// Bind the socket to the local IP address
// and port 27015
hostent* thisHost;
char* ip;
u_short port;

port = 27015;
thisHost = gethostbyname("");
ip = inet_ntoa (*(struct in_addr *)*thisHost->h_addr_list);
service.sin_family = AF_INET;
service.sin_addr.s_addr = inet_addr(ip);
service.sin_port = htons(port);
if ( bind( ListenSocket,(SOCKADDR*) &service, sizeof(service) ) ==
SOCKET_ERROR )
{ printf("bind failed\n");
closesocket(ListenSocket);
return; }
```

Opzioni del livello SOL_SOCKET: Esempio 2 (cont.)

```
// Initialize variables and call setsockopt.
// The SO_KEEPALIVE parameter is a socket option
// that makes the socket send keepalive messages
// on the session. The SO_KEEPALIVE socket option
// requires a boolean value to be passed to the
// setsockopt function. If TRUE, the socket is
// configured to send keepalive messages, if FALSE
// the socket configured to NOT send keepalive messages.
// This section of code tests the setsockopt function
// by checking the status of SO_KEEPALIVE on the socket
// using the getsockopt function.

BOOL bOptVal = TRUE;
int bOptLen = sizeof(BOOL);
int iOptVal;
int iOptLen = sizeof(int);
if (getsockopt(ListenSocket, SOL_SOCKET, SO_KEEPALIVE, (char*)&iOptVal,
&iOptLen) != SOCKET_ERROR)
        { printf("SO_KEEPALIVE Value: %ld\n", iOptVal); }
if (setsockopt(ListenSocket, SOL_SOCKET, SO_KEEPALIVE, (char*)&bOptVal,
bOptLen) != SOCKET_ERROR)
        { printf("Set SO_KEEPALIVE: ON\n"); }
if (getsockopt(ListenSocket, SOL_SOCKET, SO_KEEPALIVE, (char*)&iOptVal,
&iOptLen) != SOCKET_ERROR)
{ printf("SO_KEEPALIVE Value: %ld\n", iOptVal); } WSACleanup(); return; }
```

Opzioni del livello SOL_SOCKET: Esempio 3

```
#include <stdio.h>
#include "winsock2.h"

void main()
{ //----------------------------------------
// Declare variables WSADATA wsaData;
SOCKET ListenSocket;
sockaddr_in service;

//----------------------------------------
// Initialize Winsock
int iResult = WSAStartup( MAKEWORD(2,2), &wsaData );
if( iResult != NO_ERROR )
printf("Error at WSAStartup\n");

//----------------------------------------
// Create a listening socket
ListenSocket = socket( AF_INET, SOCK_STREAM, IPPROTO_TCP );
if (ListenSocket == INVALID_SOCKET)
{ printf("Error at socket()\n");
WSACleanup();
return; }
```

Opzioni del livello SOL_SOCKET: Esempio 3 (cont.)

```
//----------------------------------------
// Bind the socket to the local IP address
// and port 27015
hostent* thisHost;
char* ip;
u_short port;
port = 27015;
thisHost = gethostbyname("");
ip = inet_ntoa (*(struct in_addr *)*thisHost->h_addr_list);

service.sin_family = AF_INET;
service.sin_addr.s_addr = inet_addr(ip);
service.sin_port = htons(port);

if ( bind( ListenSocket,(SOCKADDR*) &service, sizeof(service) ) ==
SOCKET_ERROR )
{ printf("bind failed\n");
closesocket(ListenSocket);
return; }
```

Opzioni del livello SOL_SOCKET: Esempio 3 (cont.)

```
//----------------------------------------
// Initialize variables and call getsockopt.
// The SO_ACCEPTCONN parameter is a socket option
// that tells the function to check whether the
// socket has been put in listening mode or not.
// The various socket options return different
// information about the socket. This call should
// return 0 to the optVal parameter, since the socket
// is not in listening mode.
int optVal;
int optLen = sizeof(int);
if (getsockopt(ListenSocket, SOL_SOCKET, SO_ACCEPTCONN, (char*)&optVal,
&optLen) != SOCKET_ERROR)
printf("SockOpt Value: %ld\n", optVal);
//----------------------------------------
// Put the listening socket in listening mode.
if (listen( ListenSocket, 100 ) == SOCKET_ERROR)
{ printf("error listening\n"); }
//----------------------------------------
// Call getsockopt again to verify that
// the socket is in listening mode.
if (getsockopt(ListenSocket, SOL_SOCKET, SO_ACCEPTCONN, (char*)&optVal,
&optLen) != SOCKET_ERROR)
printf("SockOpt Value: %ld\n", optVal); WSACleanup();
return; }
```

Opzioni per le Socket: Opzioni del livello IPPROTO_TCP

TCP_NODELAY Per default il TCP abilita il cosiddetto Algoritmo di Nagle (Nagle Algorithm) il cui scopo e' di diminuire il numero di segmenti piccoli trasmessi, come nel caso di client telnet che prevedono l'ACK per ogni singolo carattere battuto a tastiera. Questo algoritmo è legato all'algoritmo dell'ACK ritardato (delayed ACK algorithm) che fa aspettare il TCP per circa 50-200 msec) prima di dare l'ack ad un segmento, sperando di poter accodare l'ACK ad un segmento di dati.

Questi due algoritmi assieme cercano di minimizzare il numero di segmenti trasmessi, ma producono ritardo per applicazioni che scambiano piccoli dati e quasi solo in una direzione.

L'opzione TCP_NODELAY disabilita l'uso di questi algoritmi.