SAPIENZA
UNIVERSITÀ DI ROMA

Corso di Laurea Specialistica in Ingegneria Informatica

a.a. 2006-2007

Tesina Metodi Formali nell'Ingegneria del Software

Verifica formale del TCP e studio di possibili attacchi usando NuSMV

Autore: Cristiano Sticca

TRANSMISSION CONTROL PROTOCOL

•TCP è un protocollo **orientato alla connessione**

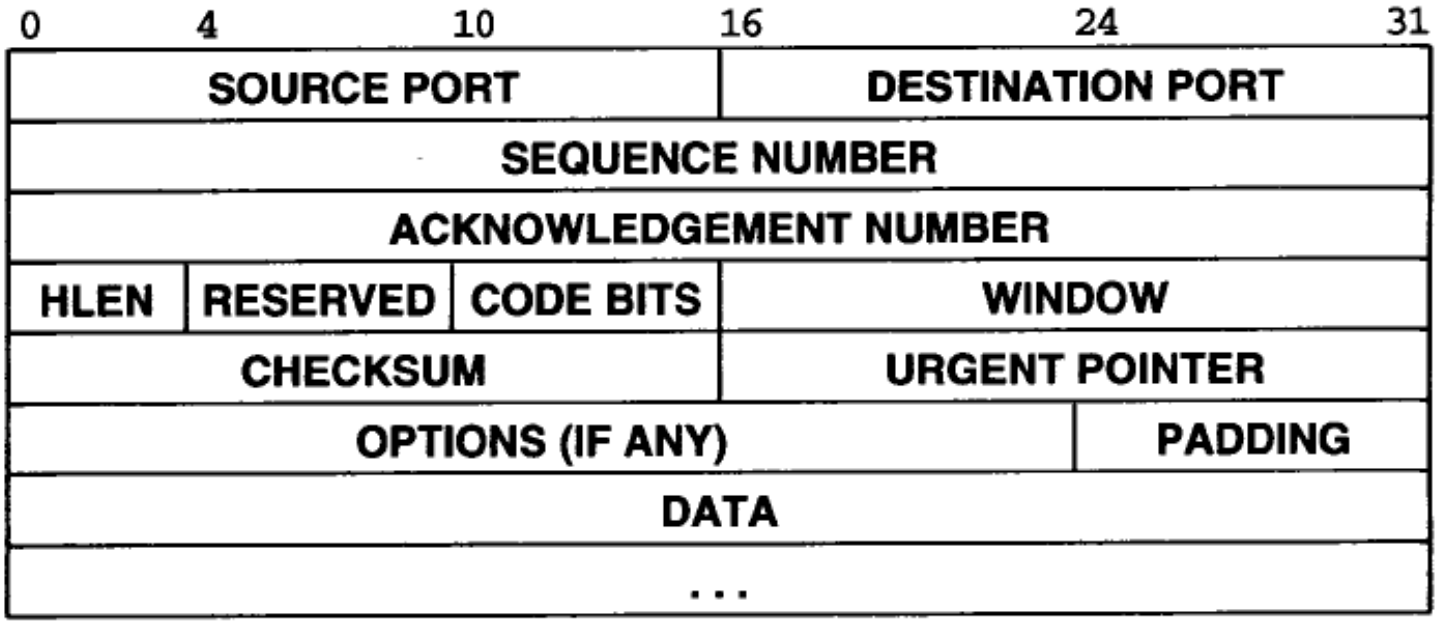
•Ha quindi le funzionalità per creare, mantenere e chiudere una connessione.

• TCP garantisce che i dati trasmessi, se giungono a destinazione, lo facciano in ordine e una volta sola.

•TCP possiede funzionalità di controllo di flusso e di controllo della congestione sulla connessione

HEADER TCP

La PDU di TCP è detta segmento. Ciascun segmento viene normalmente "imbustato" in un pacchetto IP, ed è costituito dall'intestazione(header) TCP e da un carico utile, ovvero dati di livello applicativo.
Un segmento TCP è così strutturato:



Code bits [6 bit] - Bit utilizzati per il controllo del protocollo:

• **URG** - se settato a 1 indica che nel flusso sono presenti *dati urgenti* alla posizione (offset) indicata dal campo *Urgent pointer*;

• **ACK** - se settato a 1 indica che il campo *Acknowledgment number* è valido;

• **PSH** - se settato a 1 indica che i dati in arrivo non devono essere bufferizzati ma passati subito ai livelli superiori dell'applicazione;

• **RST** - se settato a 1 ripristina la connessione; viene utilizzato in caso di grave errore;

• **SYN** - se settato a 1 indica che l'host mittente del segmento vuole *aprire una connessione TCP* con l'host destinatario e specifica nel campo *Sequence number* il valore dell' *Initial Sequence Number* (*ISN*); ha lo scopo di sincronizzare i numeri di sequenza dei due host. L'host che ha inviato il SYN deve attendere dall'host remoto un pacchetto SYN/ACK.

• **FIN** - se settato a 1 indica che l'host mittente del segmento vuole *chiudere la connessione TCP* aperta con l'host destinatario. Il mittente attende la conferma dal ricevente (con un FIN-ACK). A questo punto la connessione è ritenuta chiusa per metà: l'host che ha inviato FIN non potrà più inviare dati, mentre l'altro host ha il canale di comunicazione ancora disponibile. Quando anche l'altro host invierà il pacchetto con FIN impostato la connessione, dopo il relativo FIN-ACK, sarà considerata completamente chiusa.

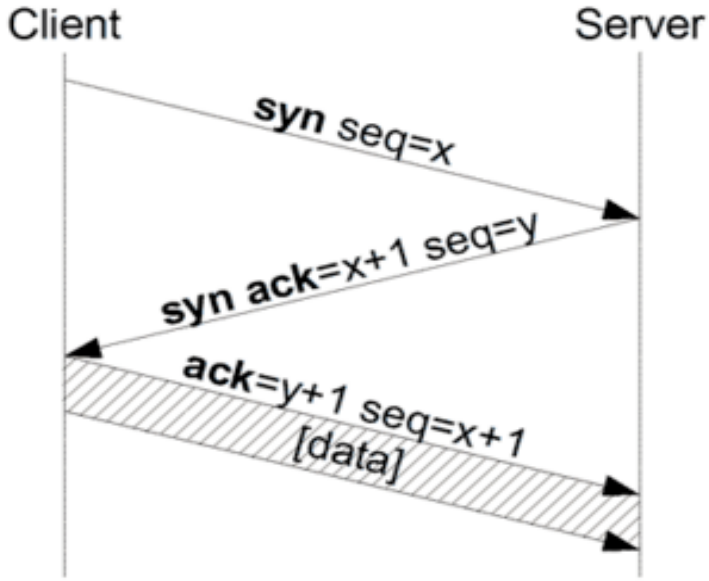
SEQUENCE NUMBER & ACKNOWLEDGMENT NUMBER

• In ricezione, TCP controlla se il numero di sequenza ricevuto è quello atteso e in caso affermativo può inviare direttamente il carico utile al processo di livello applicativo e liberare i propri buffer.
• Se invece riceve un numero di sequenza maggiore di quello atteso, deduce che uno o più segmenti ad esso precedenti sono andati persi o ritardati dal livello di rete sottostante. Pertanto, memorizza temporaneamente in un buffer il carico utile del segmento per poterlo consegnare al processo applicativo dopo aver ricevuto e consegnato anche tutti quelli precedenti.
→CONSEGNA ORDINATA DEI DATI

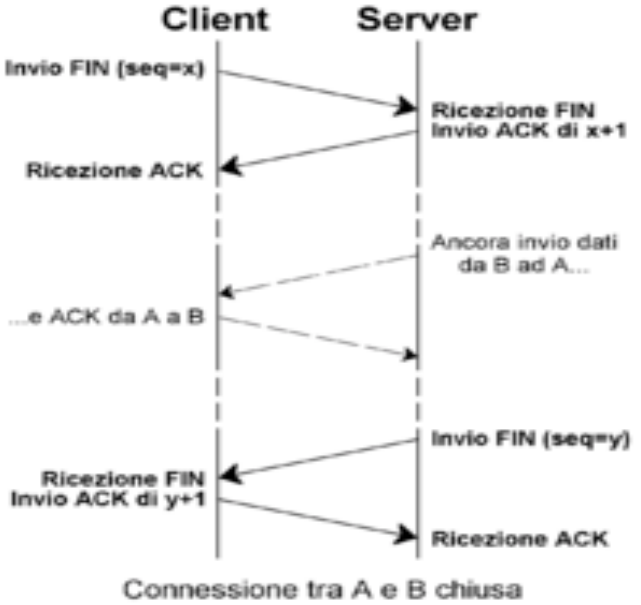
• Se infine il numero di sequenza ricevuto è inferiore a quello atteso, il segmento viene considerato un duplicato di uno già ricevuto e già inviato allo strato applicativo, e quindi scartato.
→ ELIMINAZIONE DEI DUPLICATI DI RETE

APERTURA DI UNA CONNESSIONE *THREE-WAY HANDSHAKE*

1. **A invia un segmento SYN a B** - il *flag* *SYN* è impostato a 1 e il campo *Sequence number* contiene il valore *x* che specifica l' *Initial Sequence Number* di A;
2. **B invia un segmento SYN/ACK ad A** - i *flag* *SYN* e *ACK* sono impostati a 1, il campo *Sequence number* contiene il valore *y* che specifica l' *Initial Sequence Number* di B e il campo *Acknowledgment number* contiene il valore *x+1* confermando la ricezione del ISN di A;
3. **A invia un segmento ACK a B** - il campo *Acknowledgment number* contiene il valore *y+1* confermando la ricezione del ISN di B.



CHIUSURA DI UNA CONNESSIONE
CHIUSURA A 4 VIE



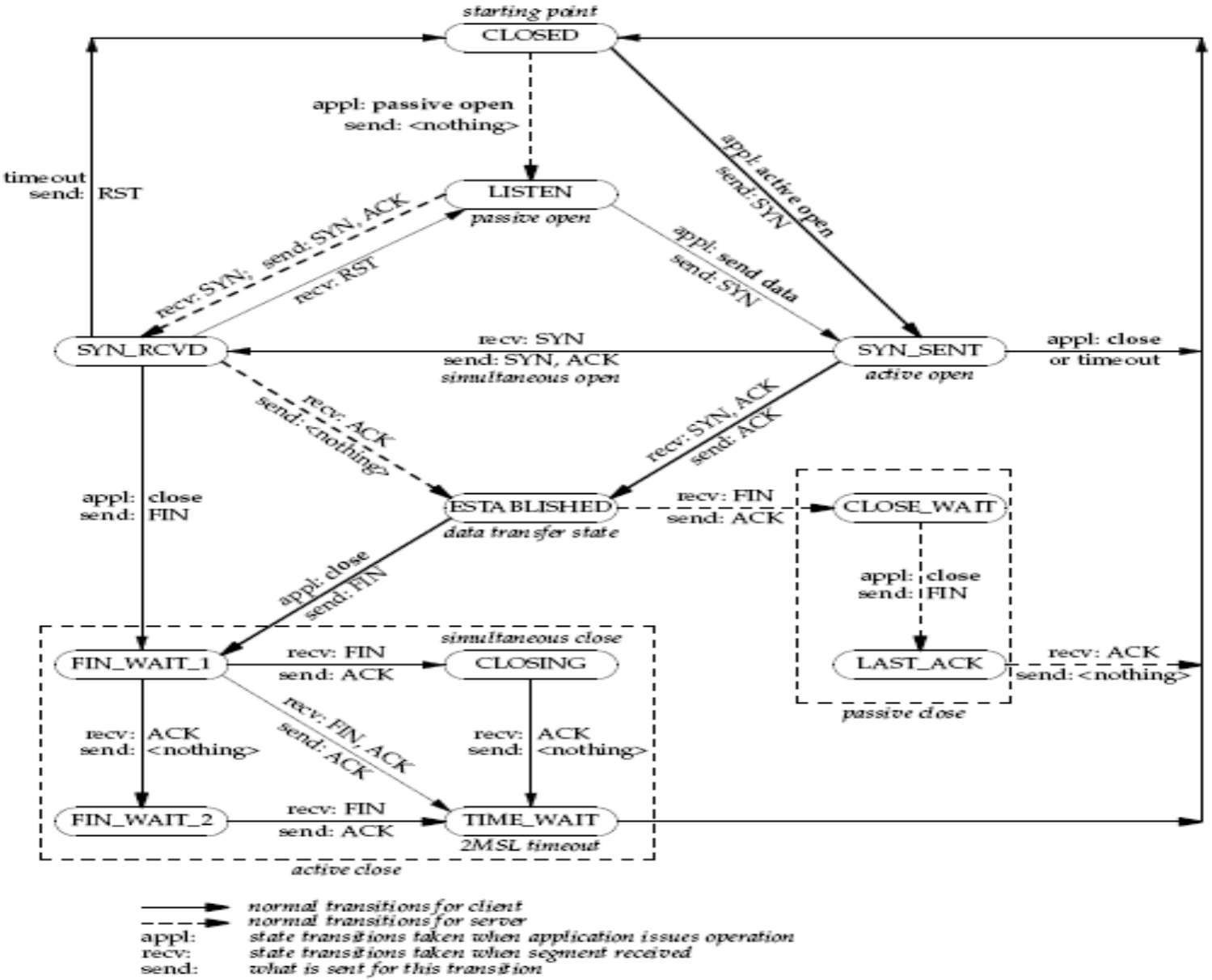
La chiusura di una connessione può essere effettuata in due modi:

- con un ***handshake a 3 vie*** in cui le due parti chiudono contemporaneamente le rispettive connessioni;
- con un ***handshake a 4 vie*** in cui le due connessioni vengono chiuse in tempi diversi

L'**handshake a 3 vie** è omologo a quello usato per l'apertura della connessione, con la differenza che il flag utilizzato è il FIN invece del SYN. Un terminale invia un pacchetto con la richiesta FIN, l'altro risponde con un FIN + ACK, ed infine il primo manda l'ultimo ACK, e l'intera connessione viene terminata.

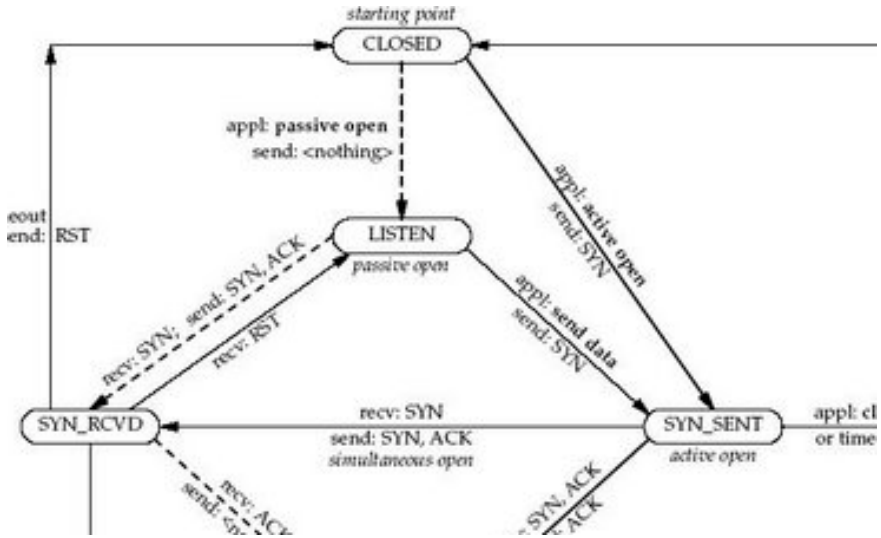
L'**handshake a 4** vie invece viene utilizzato quando la disconnessione non è contemporanea tra i due terminali in comunicazione. In questo caso uno dei due terminali invia la richiesta di FIN, e attende l'ACK. L'altro terminale farà poi altrettanto, generando quindi un totale di 4 pacchetti.

TCP STATE MACHINE



CLOSED

•stato iniziale
•per uscire da questo stato si deve fare un'operazione di open
  •open passiva non manda nulla e passa allo stato LISTEN
  •open attiva spedisce un messaggio SYN e passa allo stato SYN-SENT



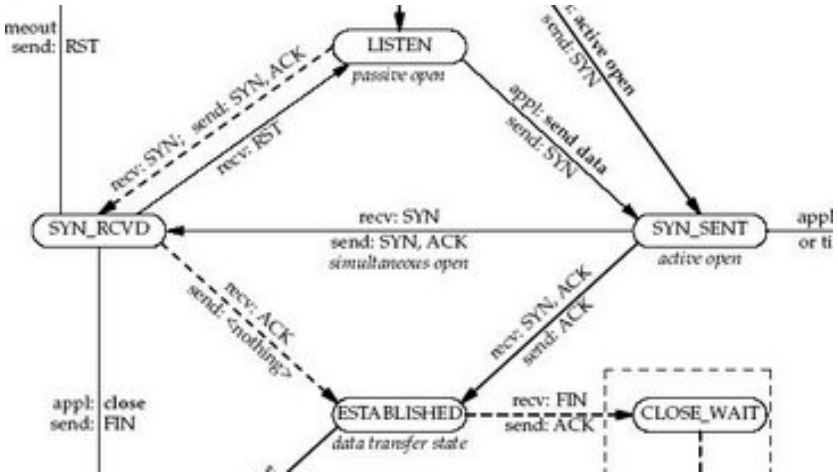
LISTEN

•in questo stato il protocollo è attivo e in ascolto su una porta
•quando riceve un SYN risponde con un SYN+ACK e passa allo stato SYN-RECEIVED
•se l'applicazione chiede di inviare dati manda un SYN e passa allo stato SYN-SENT

SYN-SENT

•stato in cui si è mandato un SYN e si attende l'ACK corrispondente
- •raggiunto da CLOSED con una open attiva o da LISTEN dopo una operazione di SEND

•attende la risposta SYN per un certo tempo
- •se riceve un SYN con ACK passa allo stato ESTABLISHED e manda a sua volta un ACK
- •se riceve un SYN senza ACK manda un SYN+ACK e passa allo stato SYN-RECEIVED
- •se non riceve risposta effettua una *close* o una *reset*



SYN-RECEIVED

•stato in cui si è ricevuto un SYN
- •se lo rifiuta, ritorna allo stato LISTEN con *reset*
- •se accetta, passa allo stato ESTABLISHED e manda l'ACK

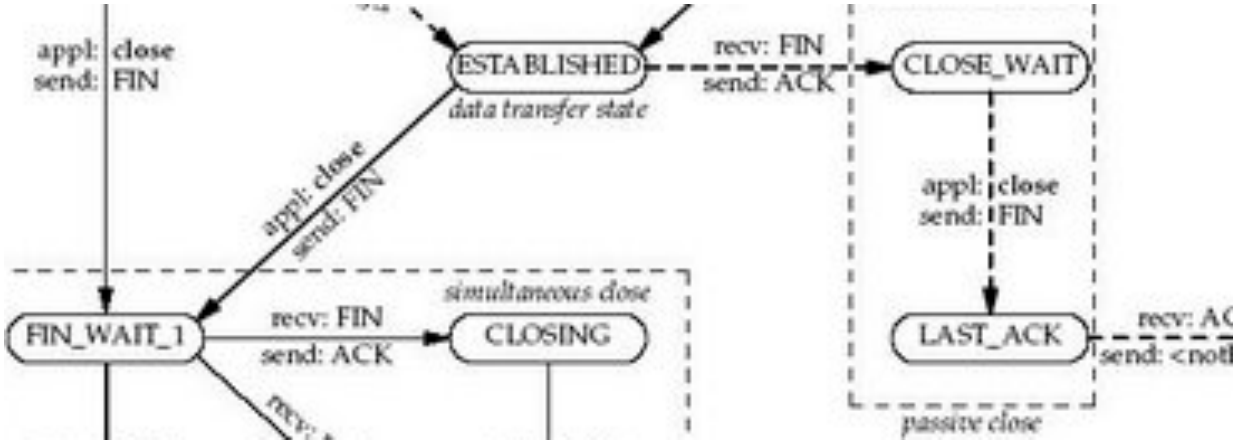
ESTABLISHED

•stato in cui è stata stabilita la connessione ed è possibile iniziare il trasferimento dati

- è stata completata la *3-way handshake*

•se l'applicazione decide di chiudere la connessione manda un messaggio di FIN e passa allo stato FIN-WAIT-1

•se riceve un messaggio FIN risponde con un ACK e passa allo stato CLOSE-WAIT



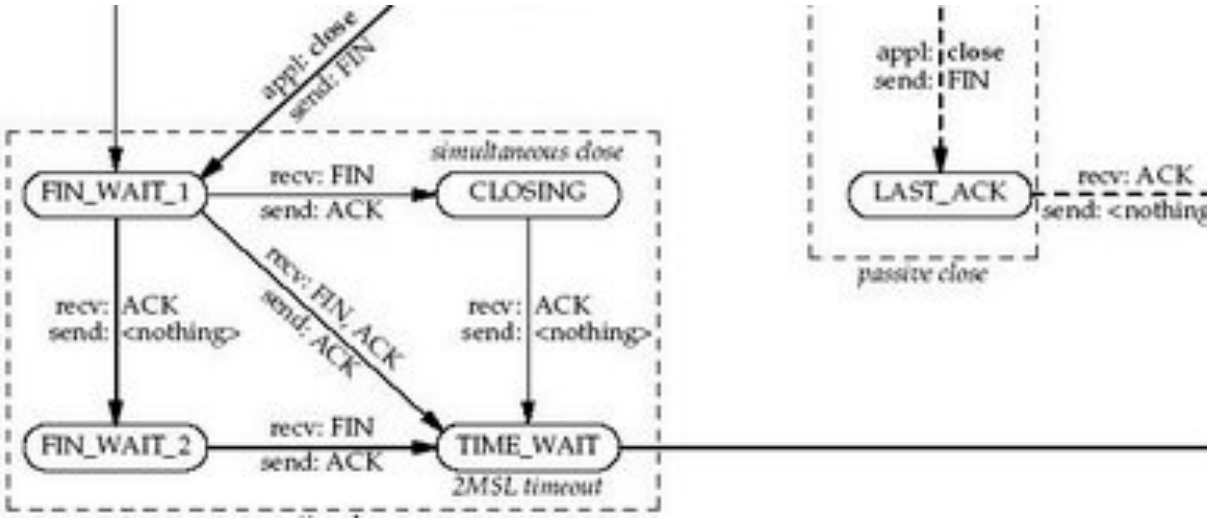
CLOSE-WAIT

•stato in cui si è ricevuto un messaggio di FIN e si attende che l'applicazione chiuda la connessione

•quando l'applicazione decidere di chiudere la connessione manda un messagio di FIN e passa allo stato LAST-ACK

LAST-ACK

•stato in cui si è ricevuto il FIN dall'altro endpoint e si è risposto con un FIN
  •Il protocollo attende l'ACK al suo FIN
•quando riceve l'ACK risponde con l'ultimo ACK e chiude la connessione



FIN-WAIT-1

•stato in cui si è inviato un messaggio FIN e si attende che l'altro endpoint chiuda la connessione
•se riceve un FIN+ACK manda l'ACK e passa allo stato TIME-WAIT
•se riceve solo un FIN manda l'ACK e passa allo stato CLOSING
•se riceve un ACK passa allo stato FIN-WAIT-2

CLOSING

•stato in cui entrambi gli endpoint hanno mandato un FIN contemporaneamente
•manda l'ACK e passa allo stato TIME-WAIT

FIN-WAIT-2

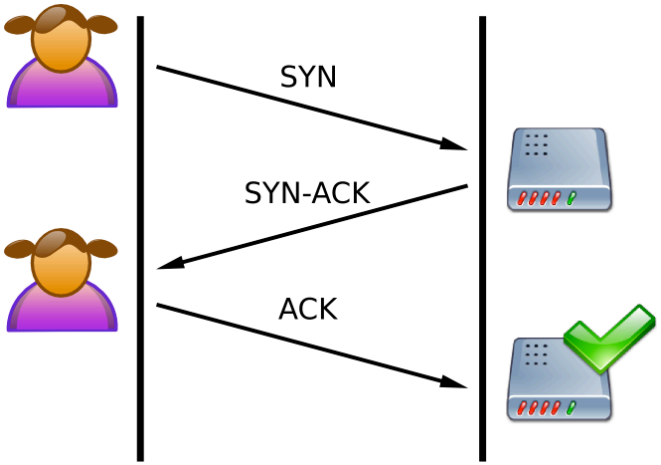
•stato in cui si è inviato un messaggio FIN per il quale è stato ricevuto l'ACK e si attende il FIN dell'altro endpoint
•quando riceve un FIN manda l'ACK e passa allo stato TIME-WAIT

TIME-WAIT

•Attende un tempo pari a 2 * MSL (maximum segment life) prima di chiudere la connessione per attendere eventuali richieste di ritrasmissione dell'ultimo ACK
•Per tutto questo intervallo di tempo la porta dell'endpoint non è utilizzabile

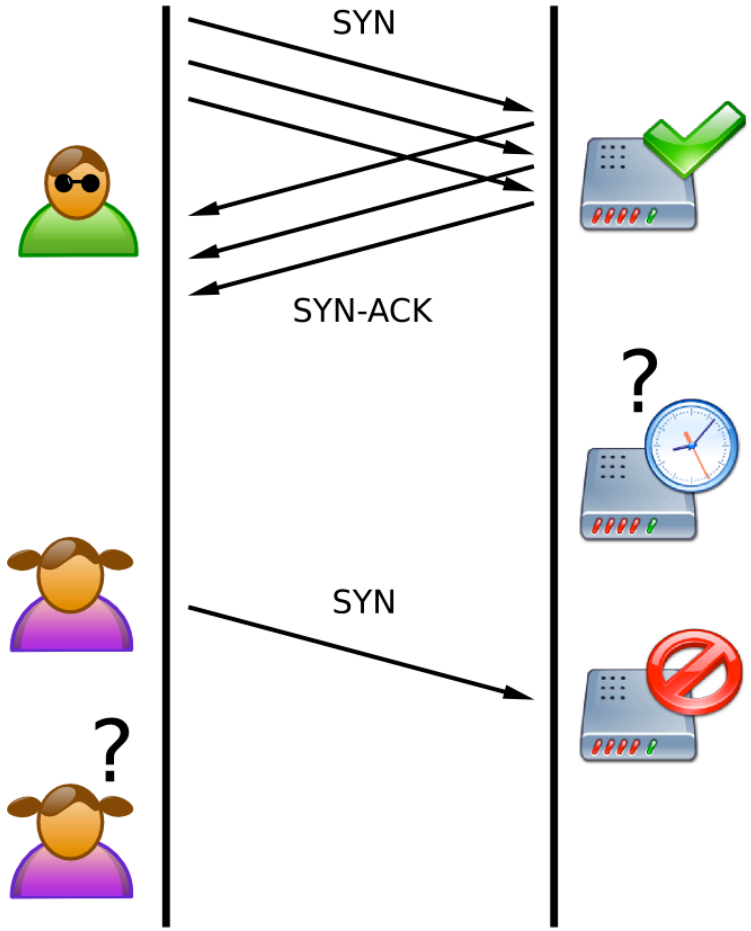
ATTACCHI PER IL TCP

→ Soprattutto attacchi DoS (denial-of-service)



Una normale connessione tra un'utente e un server. L'handshake in tre fasi avviene correttamente.

TCP SYN FLOODING



SYN flood. Un attacco viene compiuto da un utente malevolo che invia diversi pacchetti ma non ritorna il segnale "ACK" al server. Le connessioni sono perciò stabilite solo in parte e utilizzano risorse del server. L'utente che vorrebbe legittimamente connettersi al server, non riesce dal momento che il server rifiuta di aprire una nuova connessione, realizzando così un attacco *denial of service*.

PROGETTO E IMPLEMENTAZIONE

OBIETTIVO

Poiché l'obiettivo primario è di trovare possibili modi per attaccare TCP, abbiamo bisogno di conoscere come inviare i giusti segmenti per cambiare lo stato della connessione. Quindi ci concentriamo nella modellazione di come arrivano i segmenti, usercalls e timeouts, e come questi ultimi cambiano lo stato della connessione cioè modelliamo gli input e le transizioni tra gli stati tralasciando gli output fatta eccezione per il reset.

GLI EVENTI

L'attività del TCP può essere caratterizzata come quella di rispondere ad eventi

USERCALL

USERCALLS

OPEN-P

OPEN-A

SEND

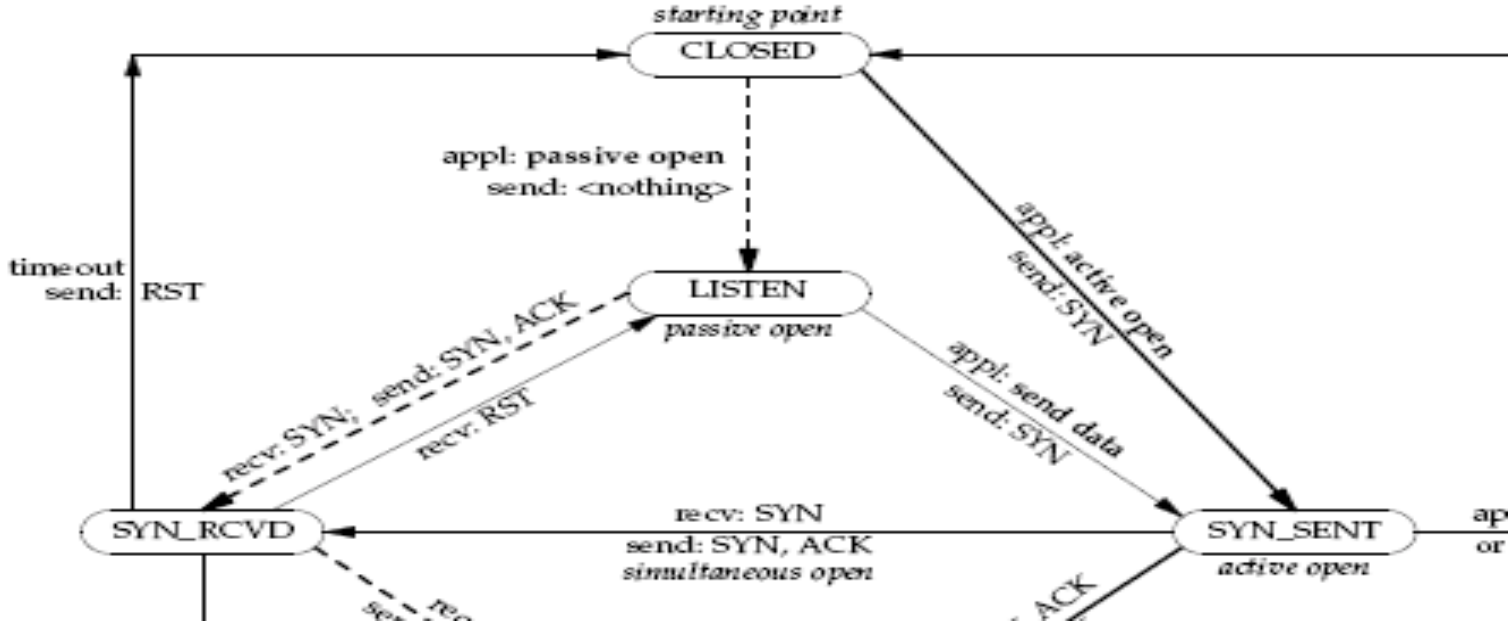
RECEIVE

CLOSE

ABORT

ACTIVE_FLAG

active_flag: boolean → **PERCHE'?**



La ragione per cui abbiamo bisogno di questa variabile è perché quando lo stato è SYN-RECEIVED e un segmento contenente un reset control bit arriva lo stato potrebbe cambiare in accordo alla precedente OPEN. Se la connessione è stata iniziata con una OPEN passiva (cioè proveniente dallo stato LISTEN), allora TCP deve ritornare allo stato LISTEN. Se invece la connessione è stata iniziata con una OPEN attiva (proveniente cioè dallo stato SYN-SENT) allora la connessione viene rifiutata

SEGMENT

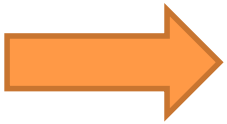
seq_ok: boolean;	**Se il sequence number è accettabile**
ack_ok: boolean;	**Se l'ACK number è accettabile**

prc_flag: {LOW, EQUAL, HIGH};	**SEG.PRG**
urg_flag: boolean;	**URG control bit**
ack_flag: boolean;	**ACK control bit**
psh_flag: boolean;	**PSH control bit**
rst_flag: boolean;	**RST control bit**
syn_flag: boolean;	**SYN control bit**
fin_flag: boolean;	**FIN control bit**

Bit (left to right)	Meaning if bit set to 1
URG	Urgent pointer field is valid
ACK	Acknowledgement field is valid
PSH	This segment requests a push
RST	Reset the connection
SYN	Synchronize sequence numbers
FIN	Sender has reached end of its byte stream

TIMEOUT

I l timeout di interesse è lo USER-TIMEOUT, in quanto gli altri necessitano della conoscenza dello scorrere del tempo.

TIMEOUT  **USER-TIMEOUT,**
RETRANSMISSION-TIMEOUT,
TIMEWAIT-TIMEOUT

GLI STATI

La variabile state può assumere tutti gli stati in accordo al diagramma degli stati e transizioni, quindi:

State → {LISTEN, SYN-SENT, SYN-RECEIVED, ESTABLISHED, FIN-WAIT-1, FIN-WAIT-2, CLOSE-WAIT, CLOSING, LAST-ACK, TIME-WAIT, CLOSED}

LA LEGGE DI POSTEL

E' raccomandato nel modellare il TCP di seguire la legge di Postel, vale a dire il principio di robustezza:

"be conservative in what you do, be liberal in what you accept from others".

→ Settiamo quindi tutti gli input in maniera non deterministica EX. next(syn_flag):{0,1}

La variabile ***active_flag*** non può essere settata nondeterministicamente poiché legata alla usercall OPEN

```
next(active_flag) := case
        event = USERCALL & usercall = OPEN-A: 1;
        event = USERCALL & usercall= OPEN-P: 0;
        1: active_flag;
esac;
```

TRANSIZIONI TRA STATI

Trattiamo i tre eventi (usercall, segment, timeout) in maniera separata e processiamo uno alla volta ognuno di questi eventi. Per ogni stato processiamo il segmento validando tutti i flag del segmento,

Ci troviamo nello stato SYN-SENT e arriva un segmento

```
next(state) := case
…
state = SYN-SENT:
                case
                                …
event = SEGMENT:
case
                ack_flag & !ack_ok: SYN-SENT;
                rst_flag & ack_flag & ack_ok: CLOSE
                rst_flag & !ack_ok: SYN-SE
                !(prc_flag = EQUAL) : SYN-
                syn_flag & ack_ok: ESTABL
                syn_flag & !ack_ok:
                1: SYN-SENT;
esac;
…
```

Con

è

to contiene il flag
zazione SYN ma
ment number
o il nuovo stato
RECEIVED

Per tutti gli altri casi non contemplati lo stato non cambia

PROCESSAMENTO DELLE **USERCALLS** E DEI **TIMEOUTS**

Il processamento delle **usercalls** è relativamente semplice. Per ogni usercall assegniamo un valore della variabile state in accordo allo stato corrente della connessione.

```
next(state):= case
        state = SYN-SENT :
            event = USERCALL:
                case
                    usercall = SEND: SYN-SENT;
usercall = CLOSE: CLOSED;
                    usercall = ABORT: CLOSED;
                    1 : SYN-SENT;
                esac;
```

Il processamento dei **timeout** è altrettanto semplice. Trattiamo solo il caso di **USER TIMEOUT**. Quando questo tipo di timeout si verifica lo stato della connessione passa a CLOSED.

```
event = TIMEOUT:
        case
        timeout = USER-TIMEOUT: CLOSED;
        1: state;
        esac;
```

PROCESSAMENTO DEI **RESETS**

La ragione per cui inseriamo nel nostro modello il reset è perché "il reset è inviato" significa che il segmento che arriva è errato oppure che la connessione sarà chiusa. Ciò è utile quando vogliamo determinare che tipo di segmento dovremmo inviare per ottenere una giusta risposta.
Analizziamo tre casi:

1. Se la connessione non esiste (CLOSED) allora il reset è inviato in risposta a qualsiasi segmento entrante eccetto un altro reset.

out_rst := case …
state = CLOSED & !rst_flag: 1

2. Se la connessione è in un altro stato non sicronizzato (LISTEN, SYN-SENT, SYN-RECEIVED) e il segmento che arriva valida qualcosa che ancora non è stato inviato (porta un ACK non accettabile) oppure il blocco di sicurezza è incorretto allora un reset è inviato.

out_rst := case …
(state = LISTEN | state = SYN-SENT | state = SYN-RECEIVED) &
((ack_flag & !ack_ok) | !(prc_flag = EQUAL)) : 1

3. Se la connessione è in uno stato sincronizzato (ESTABLISHED, FIN-WAIT-1, FIN-WAIT-2, CLOSE-WAIT, CLOSING, LAST-ACK, TIME-WAIT), per ogni segmento non accettabile (sequence number errato oppure ACK non accettabile) viene inviato un reset. Questo viene inviato anche se il campo di sicurezza/precedenza risulta essere errato.

out_rst := case …
(state = ESTABLISHED | state = FIN-WAIT-1 | state = FIN-WAIT-2 | state = CLOSE-WAIT | state = CLOSING | state = LAST-ACK | state = TIME-WAIT) & (!seq_ok | (ack_flag & !ack_ok) | !(prc_flag=EQUAL)) : 1

VERIFICA DEL SISTEMA E ANALISI ATTACCHI

CONSISTENZA

Il nostro obiettivo finale è quello di trovare possibili modi di attaccare TCP quindi tutte le verifiche e gli sfruttamenti devono essere fatte su un modello consistente.

Ci sono 22 transizioni nella nostra TCP state machine.

Prendiamo in esame 4 transizioni verificate

- LISTEN → SYN-RECEIVED
- LISTEN → SYN-SENT
- SYN-SENT → ESTABLISHED
- SYN-RECEIVED → ESTABLISHED

La ragione per cui ho preso in considerazione queste transizioni è dovuta al fatto che ci sono solo due stati (SYN-SENT, SYN-RECEIVED) che portano allo stato ESTABLISHED. Quindi questi due stati sono obiettivi di un possibile attacco. Inoltre è stato preso in considerazione lo stato LISTEN in quanto è il primo stato che si incontra nello stabilire una connessione.

ANALISI DELLE TRANSIZIONI

LISTEN → SYN-RECEIVED

EF (state = LISTEN & event = SEGMENT & !rst_flag & !ack_flag & syn_flag & (prc_flag = EQUAL))

AG ((state = LISTEN & event = SEGMENT & !rst_flag & !ack_flag & syn_flag & (prc_flag = EQUAL)) → AX (state = SYN-RECEIVED))

Questa proprietà certifica che se il TCP è nello stato LISTEN e un segmento che contiene il SYN control bit e non contiene ACK e RST bit ed ha un corretto livello di precedenza, allora il prossimo stato è SYN-RECEIVED.

LISTEN → SYN-SENT

EF (state = LISTEN & event = USERCALL & usercall = SEND)

AG ((state = LISTEN & event = USERCALL & usercall = SEND) → AX(state= SYN-SENT))

La proprietà sopra citata illustra che se lo stato è LISTEN ed il prossimo evento è una usercall del tipo SEND il prossimo stato deve essere SYN-SENT.

SYN-SENT → ESTABLISHED

EF (state = SYN-SENT & event = SEGMENT & ack_flag & ack_ok & !rst_flag & (prc_flag = EQUAL) & syn_flag)

AG((state = SYN-SENT & event = SEGMENT & ack_flag & ack_ok & !rst_flag & (prc_flag = EQUAL) & syn_flag) → AX(state = ESTABLISHED))

Se lo stato è SYN-SENT e si hanno le seguenti condizioni:
se il segmento contiene un acknowledgment corretto ed un ACK control bit, se non contiene un RST bit, se ha un livello di precedenza esatto ed infine se contiene il SYN flag allora il prossimo stato deve essere ESTABLISHED.

SYN-RECEIVED → ESTABLISHED

EF (state = SYN-RECEIVED & event = SEGMENT & seq_ok & !rst_flag & (prc_flag = EQUAL) & !syn_flag & ack_flag & ack_ok & !fin_flag)

AG ((state = SYN-RECEIVED & event = SEGMENT & seq_ok & !rst_flag & (prc_flag = EQUAL) & !syn_flag & ack_flag & ack_ok & !fin_flag)
→ AX(state = ESTABLISHED))

Se lo stato è SYN-RECEIVED e si hanno le seguenti condizioni: se il segmento contiene un corretto sequence number, se non contiene un RST bit né un SYN né un FIN, se ha un livello di precedenza esatto ed infine se contiene un ACK con il numero di acknowledgment corretto allora il prossimo stato deve essere ESTABLISHED.

TRANSIZIONI NON VERIFICABILI

Le seguenti transizioni non sono state verificate:

- FIN-WAIT-1 → CLOSING
- FIN-WAIT-1 → FIN-WAIT-2
- FIN-WAIT-1 → TIME-WAIT
- CLOSING → TIME-WAIT
- LAST-ACK → CLOSED
- ESTABLISHED → CLOSE-WAIT

Esse non sono verificabili in quanto per essere verificabili c'è necessità di mantenere l'informazione se il segmento con l'ACK è in risposta al messaggio di FIN inviato precedentemente.
Ma nel nostro modello non mantenendo informazioni circa quello che viene inviato non è possibile sapere se l'ACK è in risposta o meno al FIN.

ANALISI ATTACCHI

Si utilizzerà NuSMV per produrre un controesempio che ci mostra l'esistenza di un possibile attacco SYN flooding; in seguito si costruirà un segmento che verrà verificato sempre con NuSMV.

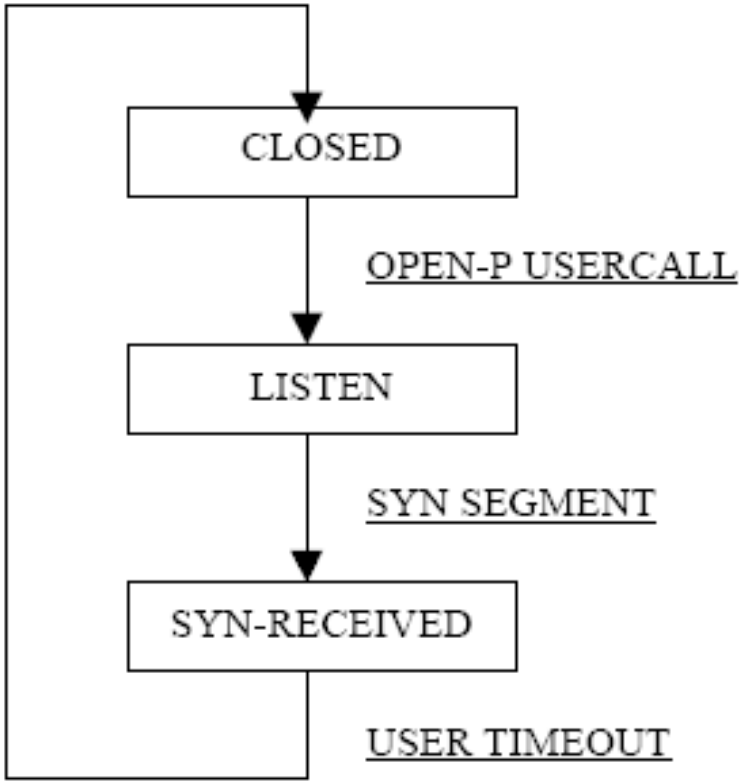
Poiché gli attacchi di tipo SYN flooding si hanno quando lo stato è in SYN-RECEIVED, si vuole verificare che se lo stato è SYN-RECEIVED allora prima o poi lo stato diverrà ESTABLISHED.

AG (state = SYN-RECEIVED → AF state = ESTABLISHED)

Inoltre poichè quando è in corso un tale attacco non ci sono ne segmenti in arrivo ne usercalls aggiungiamo un proprietà di fairness:

FAIRNESS
state = SYN-RECEIVED ->
!(event = USERCALL | event = SEGMENT)

NuSMV produce un controesempio.
Il contro esempio, illustrato nella figura seguente, assicura che se lo USER TIMEOUT avviene infinitamente spesso quando lo stato è SYN-RECEIVED allora lo stato non sarà mai ESTABLISHED, e ciò mostra la possibilità di un attacco di tipo SYN flooding.



COSTRUZIONE DEL SEGMENTO (1)

Verifichiamo dapprima che sia possibile cambiare lo stato da LISTEN a SYN-RECEIVED quando arriva un segmento SYN.
Ecco la proprietà:

EF(event = SEGMENT & state = LISTEN & syn_flag ->
AX(state = SYN-RECEIVED))

La proprietà è verificata essere vera. Ora vogliamo verificare se questo attacco possa avvenire sempre.

AG(event = SEGMENT & state = LISTEN & syn_flag & !rst_flag & !
ack_flag & prc_flag = EQUAL
→
AX(state = SYN-RECEIVED))

Anche questa proprietà risulta essere vera.

COSTRUZIONE DEL SEGMENTO (2)

Dalle proprietà verificate concludiamo che ci sono tre passi nel nostro attacco:

1.Inviare un corretto SYN segment alla vittima;
2.Non rispondere a nessun messaggio inviato dalla vittima e far si che la connessione vada in timeout.
Un modo per fare ciò è di inserire un indirizzo IP falso.
3. Quando la memoria del backlog è piena tutte le richieste di connessione vengono ignorate →

L'HOST E' SOTTO UN ATTACCO SYN FLOODING!!!

UN NUOVO ATTACCO (1)

Abbiamo notato in precedenza che un segmento con un SYN control bit può causare la chiusura della connessione.
Ci chiediamo ora se la connessione potrebbe essere chiusa quando c'è un segmento con un corretto sequence number e un SYN control bit e quando lo stato corrente non è né LISTEN né SYN-SENT.

AG(!(state = LISTEN | state = SYN-SENT) &
event = SEGMENT & seq_ok & syn_flag → AX(state = CLOSED))

Ovviamente la proprietà non è vera. Il motivo è perché il TCP controlla il RST bit e il flag di sicurezza e precedenza prima del SYN flag. Quindi la proprietà corretta è la seguente:

AG(!(state = LISTEN | state = SYN-SENT) &
event = SEGMENT & seq_ok & !rst_flag & (prc_flag = EQUAL) &
syn_flag → AX(state = CLOSED))

UN NUOVO ATTACCO (2)

Quindi il seguente potrebbe essere un attacco realistico:

1.Il cracker sa che c'è una connessione tra due host A e B
2.Il cracker vuole rompere la connessione
3.Per fare ciò deve fare in modo che A o B chiudano la connessione
4.Supponiamo che voglia far chiudere la connessione ad A. Esso falsifica l'indirizzo di B ed invia il segmento SYN ad A. In seguito A chiuderà la connessione.

Così se il cracker può scoprire e distruggere ogni connessione che l'host vittima vuole stabilire, allora l'host è sotto un attacco di tipo DoS.

La difficoltà di questo attacco risiede nel poter acquisire nella maniera esatta i numeri di sequenza. Un modo per far ciò è attraverso lo sniffing.

UN NUOVO ATTACCO (3)