Advanced Computer Networks

Ibrahim Matta

Management of Protocol State

References: some slides courtsey of Richard Yang and Jim Kurose, work in Belsnes'76, Delta-t'78, Ji et al. SIGCOMM'03, and Lui et al. ICNP'04

Maintaining protocol/network state

state: information *stored* in network nodes by network protocols

- updated when network / transfer "conditions" change
- stored in multiple nodes
- often associated with end-system generated call or session
- examples:
  - RSVP routers maintain lists of upstream sender IDs, downstream receiver reservations
  - ATM switches maintain lists of VCs: bandwidth allocations, interface/VCI input-output mappings
  - TCP: Sequence numbers, timer values, RTT estimates

Soft-state

- state *installed* by receiver on receipt of setup (trigger) msg from sender (typically, an endpoint)
  - sender also sends periodic *refresh msg*: indicating receiver should continue to maintain state
- state *removed* by receiver via timeout, in absence of refresh msg from sender
- *default assumption:* state becomes invalid unless refreshed
  - in practice: explicit state removal (*teardown*) msgs also used
- examples:
  - RSVP, RTP, IGMP, Delta-t

Hard-state

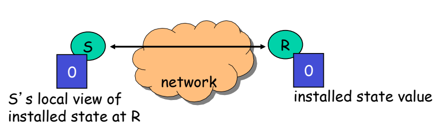
- state *installed* by receiver on receipt of *setup msg* from sender
- state *removed* by receiver on receipt of *teardown msg* from sender
- *default assumption:* state valid unless told otherwise
  - in practice: failsafe-mechanisms (to remove orphaned state) in case of sender failure, e.g., receiver-to-sender "heartbeat": is this state still valid?
- examples:
  - Q.2931 (ATM Signaling)
  - ST-II (Internet hard-state signaling)
  - TCP (explicit handshaking for opening/closing connections)

State: senders, receivers

- sender: network node that (re)generates signaling (control) msgs to install, keep-alive, remove state from other nodes
- receiver: node that creates, maintains, removes state based on signaling msgs received from sender

Let's build signaling protocol

- *S:* state *S*ender (state installer)
- *R:* state *R*eceiver (state holder)
- desired functionality:
  - S: set values in R to 1 when "installed", set to 0 when not installed
  - if other side is down, state is not installed (0)
  - initial condition: state not installed

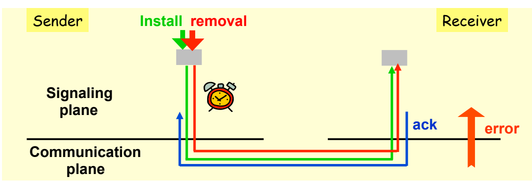


Let's build signaling protocol

Now: design and specification

Later: performance model

Hard-state signaling

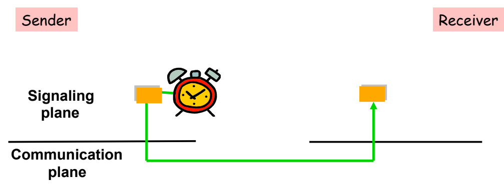


- reliable signaling
- state removal by request
- requires additional error handling
  - e.g., sender failure

Soft-state signaling

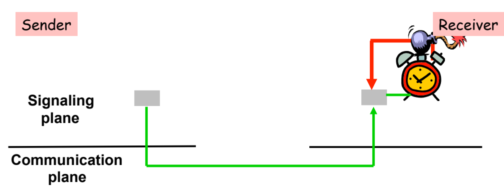
- best effort signaling

Soft-state signaling



- best effort signaling
- refresh timer, periodic refresh

Soft-state signaling



- best effort signaling
- refresh timer, periodic refresh
- state time-out timer, state removal only by time-out

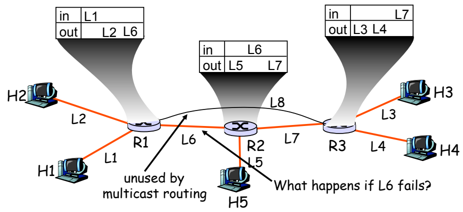
Soft-state: claims

- "Systems built on soft-state are robust" [Raman 99]
- "Soft-state protocols provide .. greater robustness to changes in the underlying network conditions…" [Sharma 97]
- "obviates the need for complex error handling software" [Balakrishnan 99]

What does this mean?

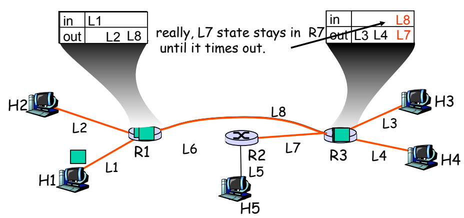
Soft-state: "easy" handling of changes

- Periodic refresh: if network "conditions" change, refresh will re-establish state under new conditions
- example: RSVP/routing interaction: if routes change (nodes fail) RSVP PATH refresh will *re-establish* state along new path



Soft-state: "easy" handling of changes

- L6 goes down, multicast routing reconfigures but…
- H1 data no longer reaches H3, H4, H5 (no sender or receiver state for L8)
- H1 refreshes PATH, establishes *new* state for L8 in R1, R3
- H4 refreshes RESV, propagates upstream to H1, establishes new receiver state for H4 in R1, R3



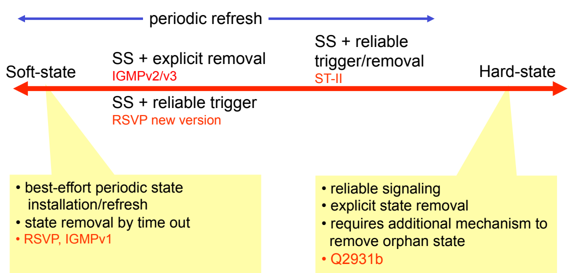
Soft-state: "easy" handling of changes

- "recovery" performed transparently to end-system by normal refresh procedures
- no need for network to signal failure/change to end system, or end system to respond to specific error
- less signaling (volume, types of messages) than hard-state from network to end-system but…
- more signaling (volume) than hard-state from end-system to network for refreshes

Soft-state: refreshes

- refresh msgs serve many purposes:
  - trigger: first time state-installation
  - refresh: refresh state known to exist ("I am still here")
  - <lack of refresh>: remove state ("I am gone")
- challenge: all refresh msgs unreliable
  - would like triggers to result in state-installation asap
  - enhancement: add receiver-to-sender refresh_ACK for triggers
  - e.g., see "Staged Refresh Timers for RSVP"

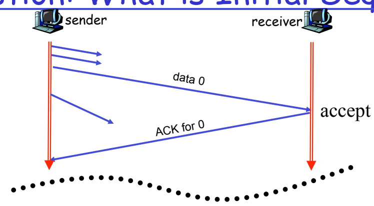
Signaling spectrum



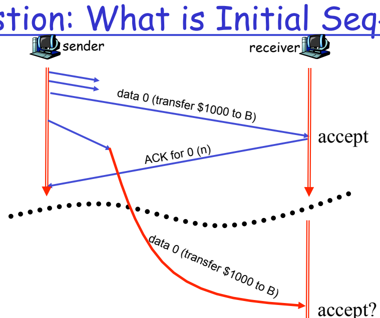
Reliable Transport

- Goal: keep states, e.g. sequence numbers sent & received, consistent to ensure correctness
  - No data loss
  - No duplication
  - In-order delivery

Question: What is Initial Seq#?



Question: What is Initial Seq#?

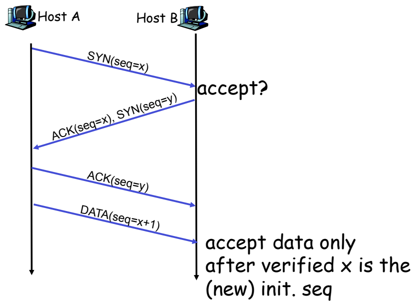


- To distinguish new data, a sender should *not* reuse a seq# before it is sure the packet has left the network

Connection Management: Objective

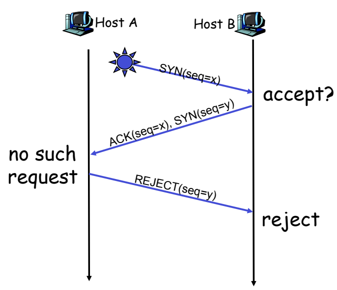
- Agree on initial sequence numbers
  - a sender will not reuse a seq# before it is sure that all packets with the seq# are purged from the network
    - the network guarantees that a packet too old will be purged from the network: network bounds the life time of each packet (MPL = Max Packet Lifetime)
  - To avoid waiting for the seq# to start a session, use a larger seq# space
    - needs connection setup so that the sender tells the receiver initial seq#
- Agree on other initial parameters

Three Way Handshake (TWH) [Tomlinson 1975]



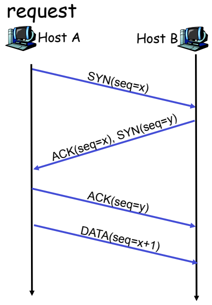
SYN: indicates connection setup

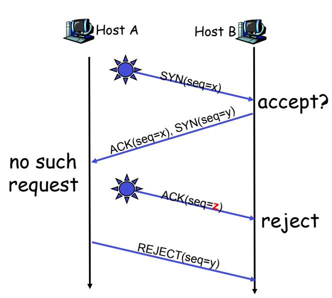
Scenarios with Duplicate Request



Three Way Handshake (TWH) [Tomlinson 1975]

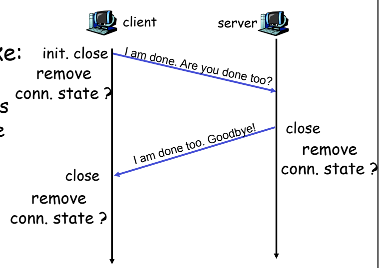
- To ensure that the other side does want to send a request





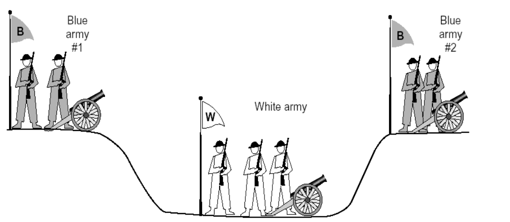
Connection Close

- Objective of closure handshake:
  - each side can release resources and remove state about the connection



25

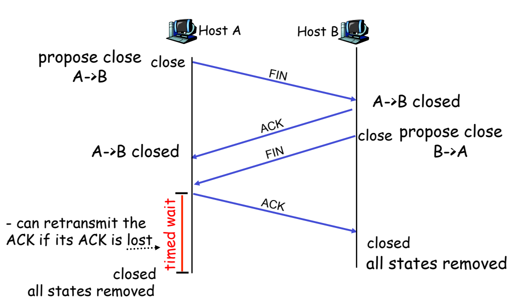
General Case: The Two-Army Problem



The two blue armies need to agree on whether or not they will attack the white army. They achieve agreement by sending messengers to the other side. If they both agree, attack; otherwise, no. Note that a messenger can be captured!

26

Four Way Teardown



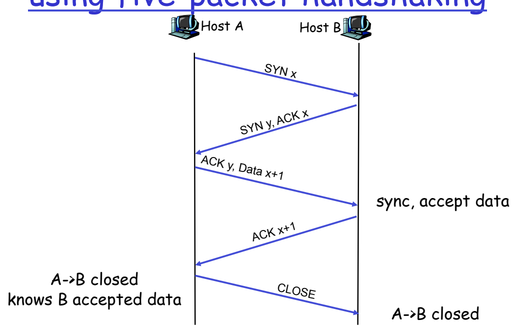
27

A Summary of Questions

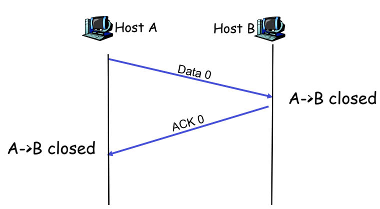
- What if there are duplication and reordering?
  - network guarantee: max packet life time (MPL)
  - transport guarantee: not reuse a seq# before life time
  - seq# / connection management
- How to determine the "right" parameters, e.g., for "timed wait"?
- What if we want to reliably send one message? (worst-case)

28

Reliable One-Message Delivery using five-packet handshaking

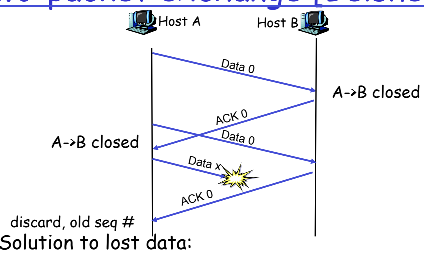


Two-packet exchange [Belsnes 76]



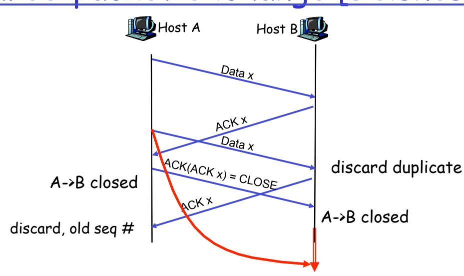
- Premature timeout results in duplicate
- Duplicate ACK may (falsely) ACK a lost "new Data 0"

Two-packet exchange [Belsnes 76]



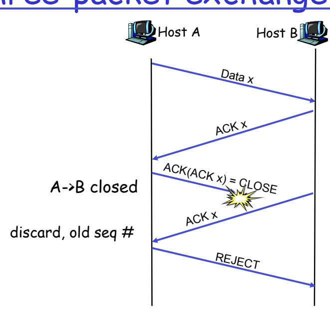
- Solution to lost data:
  use a new seq # that does NOT wrap around for at least 2 * MPL (Max Packet Lifetime)
- Duplicates still possible if ACK is lost, even with RTO > 2 * MPL

Three-packet exchange [Belsnes 76]



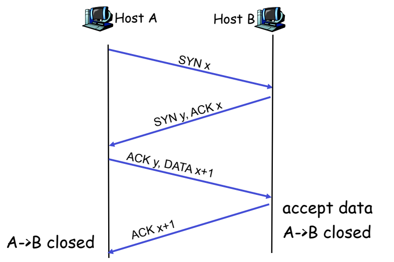
- What if retransmitted “Data x” comes in late?
  If B accepts it, duplicate; if B rejects it, might be new data
- Solution? B does not accept data until it gets ACK(ACK)

Three-packet exchange [Belsnes 76]



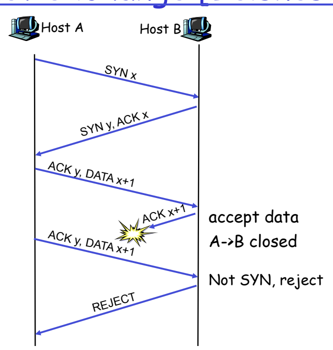
- Data lost (not yet accepted by B) if ACK(ACK) is lost
- Problem: state inconsistency, A closed while B still open

Four-packet exchange [Belsnes 76]

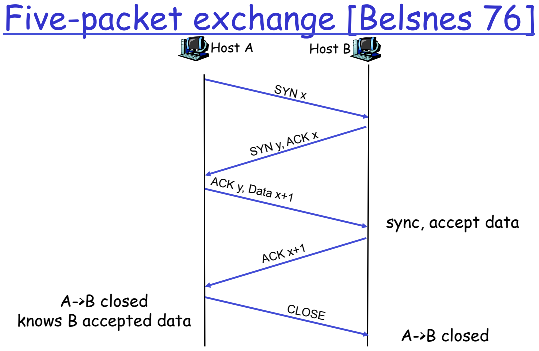


•Solution: sync both sides before accepting data
• Problem: sender does not know whether Data got accepted if last ACK is lost

Four-packet exchange [Belsnes 76]



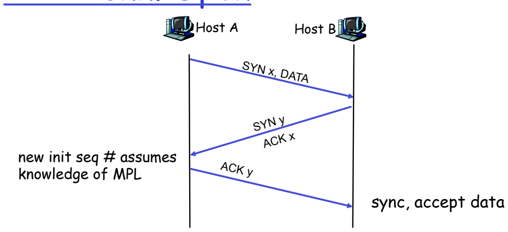
Five-packet exchange [Belsnes 76]



Moral of the story

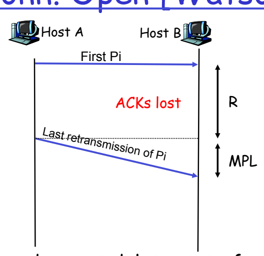
- Two-packet exchange suffices if we can leave it to applications to detect duplicates
- Delta-t solves the duplicate problem of two-packet using appropriate timers for keeping conn. state

TCP: Conn. Open



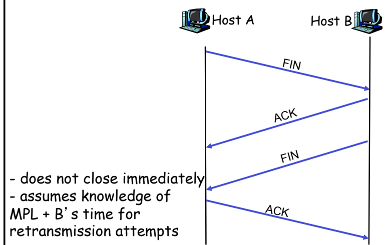
- Conn. Opening Problem: Old duplicates causes conn. to re-open & duplicates delivered

Delta-t: Conn. Open [Watson 78]



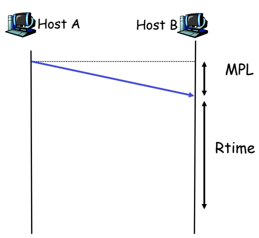
- Delta-t receiver does not delete state for at least
  Rtime = R+MPL
  enough for duplicates to die out
- R = max time for retransmission attempts
- Rtime reset at every reception of new in-seq packet

TCP: Conn. Close



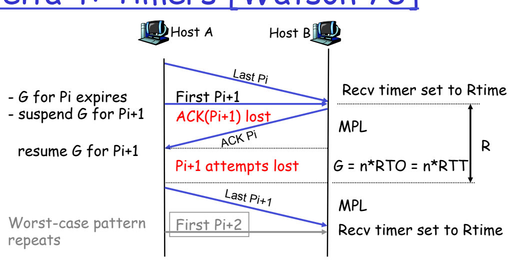
- Conn. Closing Problem: sender has to make sure that receiver got its data, including last ACK

Delta-t: Conn. Close [Watson 78]



- Delta-t sender does not delete state for at least Stime = Rtime+MPL
- Stime reset at every transmission

Delta-t: Timers [Watson 78]



- Rtime >= R + MPL = (MPL + G) + MPL ~ 2MPL, if MPL>>G
- Stime >= Rtime+MPL ~ 3MPL

Moral of the Story

- TCP is really hybrid HS+SS
  - Explicit handshaking to open/close conn.
  - We need to know something about MPL for sender to choose init seq # and to remove conn. state
- Delta-t is SS
  - No need for explicit signaling to open/close conn.
  - No need to worry about init seq # since conn. state at both sender & receiver is not removed until all its packets have died out
    - If receiver has state then conn. is not new; no need to verify with sender

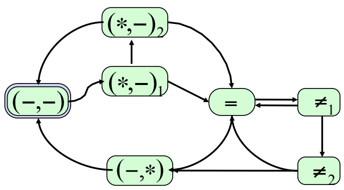
Performance & Robustness Analysis

- We looked at keeping states consistent to ensure data correctness
- Next consider a general signaling (state management) model
- Evaluate HS vs. SS analytically

Evaluation metrics

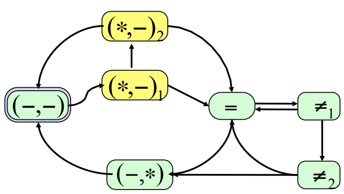
- inconsistency ratio - fraction time participating nodes disagree
- signaling overhead – average # of messages during session lifetime
- robustness? (resilience to changing conditions)
- complexity?

Performance Model for SS (Ji03)



Performance Model for SS

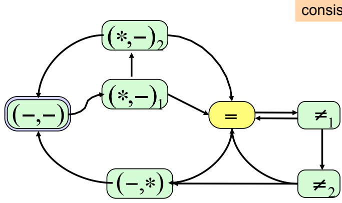
(*,–): signaling state generated at sdr, not installed at rcvr



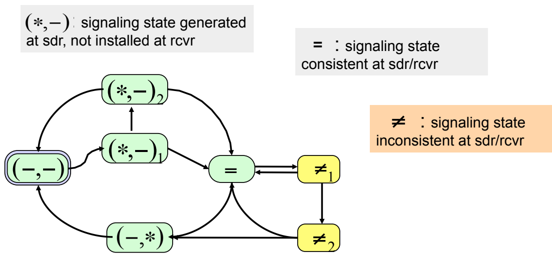
Performance Model for SS

(*,–) signaling state generated at sdr, not installed at rcvr

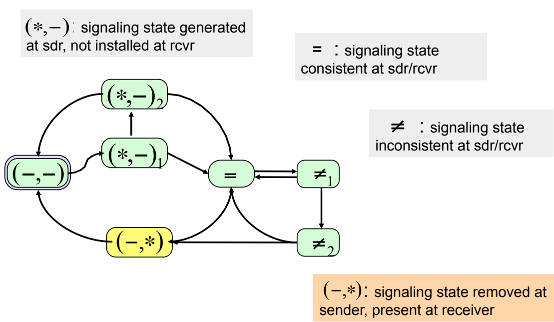
= : signaling state consistent at sdr/rcvr



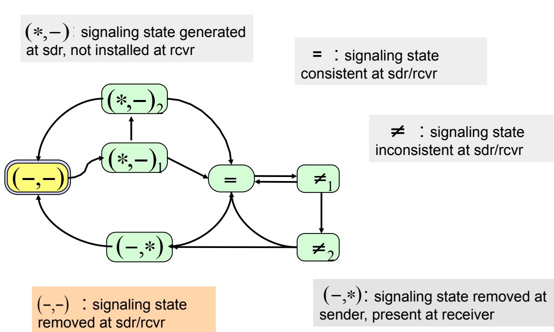
Performance Model for SS



Performance Model for SS



Performance Model for SS



Performance Model for SS

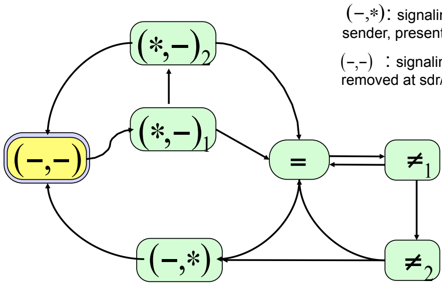
(*,–): signaling state generated at sdr, not installed at rcvr

= : signaling state consistent at sdr/rcvr

≠ : signaling state inconsistent at sdr/rcvr

(–,*): signaling state removed at sender, present at receiver

(–,–) : signaling state removed at sdr/rcvr



Performance Model for SS

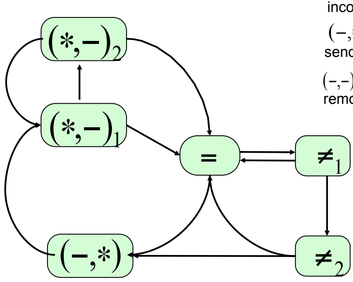
(*,–): signaling state generated at sdr, not installed at rcvr

= : signaling state consistent at sdr/rcvr

≠ : signaling state inconsistent at sdr/rcvr

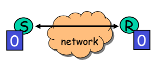
(–,*): signaling state removed at sender, present at receiver

(–,–) : signaling state removed at sdr/rcvr



Assume new session starts once previous ends

Performance Model for SS



- sender, receiver, single state variable
- events:
  - *state removal:* sender wants to remove state, mean state lifetime: $1/\mu$
  - *state update:* sender wants to change state, meantime between updates: $1/\lambda$
  - *timeouts:*
    - refresh timeout at S – mean T
    - Soft state timeout at R – mean X
  - *message arrival/loss:* mean delay D, loss prob. p

Performance Model for SS

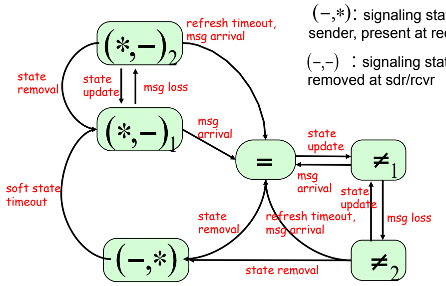
(∗,−) : signaling state generated at sdr, not installed at rcvr

= : signaling state consistent at sdr/rcvr

≠ : signaling state inconsistent at sdr/rcvr

(−,∗): signaling state removed at sender, present at receiver

(−,−) : signaling state removed at sdr/rcvr



Performance Model for SS

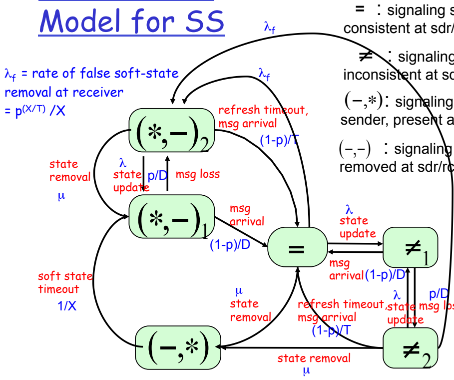
(∗,−) : signaling state generated at sdr, not installed at rcvr

= : signaling state consistent at sdr/rcvr

≠ : signaling state inconsistent at sdr/rcvr

(−,∗): signaling state removed at sender, present at receiver

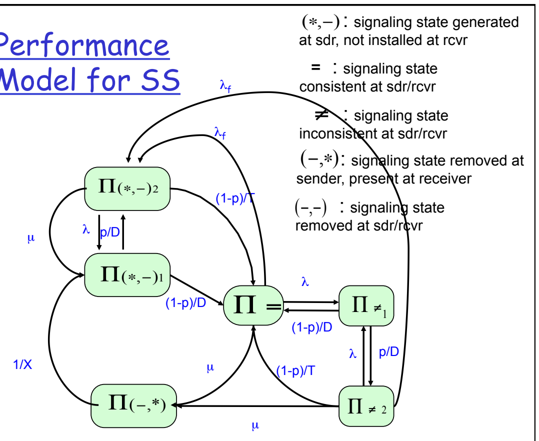
(−,−) : signaling state removed at sdr/rcvr



Performance Model for SS: analysis

- *states:* X_1, X_2 …. X_6 (six states from previous slide)
- *transition rates:* state X_i to state X_j: $\lambda_{i,j}$
  - assumption: time between transitions exponentially distributed, mean given by rates from previous slide
- *goal:* compute steady-state probability of being in state, $\pi_i = \lim_{t \to \infty} P(X(t)=i)$, i= 1,..6
- *solve system of linear equations:*
  - rate of transitions out of state = rate of transitions into state: $\Sigma_{j \neq i} \lambda_{i,j} \pi_i = \Sigma_{j \neq i} \lambda_{j,i} \pi_j$, i =1,..6
  - normalization: $\Sigma_i \pi_i = 1$

Performance Model for SS



Performance Model for SS: metrics

- inconsistency: fraction of time S, R, have different states:

$$\delta = 1 - \pi_{=}$$

- signaling overhead =

$$\sum_{\text{all states, } i} \Pi_i \cdot \text{signaling rate in state } i$$

$$= (\Pi_{(*,-)_1} + \Pi_{\neq_1})/D + (\Pi_{(*,-)_2} + \Pi_{=} + \Pi_{\neq_2})/T$$

Parameter settings

- mean lifetime – 30 min.
- refresh timer, T=5sec
- state timer, X = 15 sec
- update rate: 1/20sec
- loss rate: p = 0.02

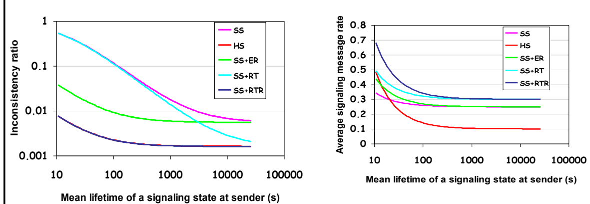
Motivated by Kazaa

Soft-state: setting timer values

Q: How to set refresh/timeout timers

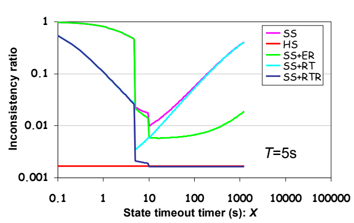
- state-timeout interval = n * refresh-interval-timeout
  - what value of n to choose?
- will determine amount of signaling traffic, responsiveness to change
  - small timers: fast response to changes, more signaling
  - long timers: slow response to changes, less signaling
- ultimately: consequence of slow/fast response, msg loss probability will dictate appropriate timer values

Impact of state lifetime



- inconsistency, overhead decrease as state life-time increases
- explicit removal improves consistency with little additional overhead

Impact of state timeout timer



- *X* < *T*: inconsistency high (premature state removal)
- *X* > *2T*: increasing *X* ⇒ increasing inconsistency for SS, SS+ER, SS+RT (due to orphan state)
- *X* = *2T*: sweet spot

Hard-state versus soft-state: discussion

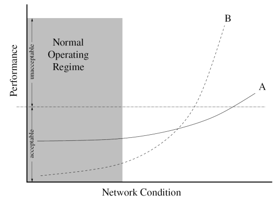
Q: which is preferable and why?

hard state:
- better if message OH really high
- potentially greater consistency
- system wide coupling -> difficult to analyze

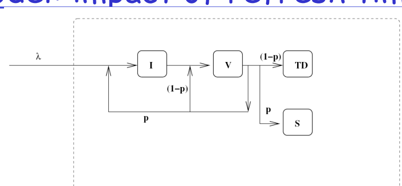
soft state:
- robustness
- easily decomposed -> simpler analysis

Robustness [Lui et al. 2004]

- Which one, A or B, is more robust?
- A is more robust since it's more resilient to unpredictable load, attacks, etc.
- Tradeoff: slightly worse performance under normal conditions



Model: impact of refresh timer



- Refresh timer = R (T in the previous model)
- State lifetime = L (1/μ in the previous model)
- State re-initialized if refresh msg is lost
- State stale (orphaned) if teardown msg is lost
- Small loss probability "p"

Costs

Cost per refresh msg
Cost of inconsistent state per unit time
Cost of re-initializing state
Cost of stale state Per unit time

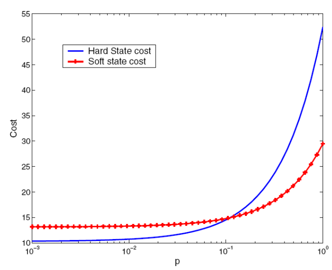
$$C(R) = C_r\frac{L}{R} + C_{is}pL + C_i(1+p\frac{L}{R}) + C_{ss}p_{ss}R$$

$$E[C(R)] = a(C_r\frac{E[L]}{R} + C_i(1+p\frac{E[L]}{R})) + C_{is}pE[L] + b(C_{ss}p_{ss}R).$$

$$R^* = \sqrt{\frac{aE[L](C_r + C_ip)}{bC_{ss}p_{ss}}}.$$

a >> b: hard state protocol – min/no refresh cost
b >> a: soft state protocol – min orphaned state cost
Optimal R* is large for HS & small for SS

Soft state more resilient



- Soft state is more resilient to increasing "p" and "L"
- SS able to overcome high loss with small R, i.e. more refreshes reduce cost by reducing stale state cost