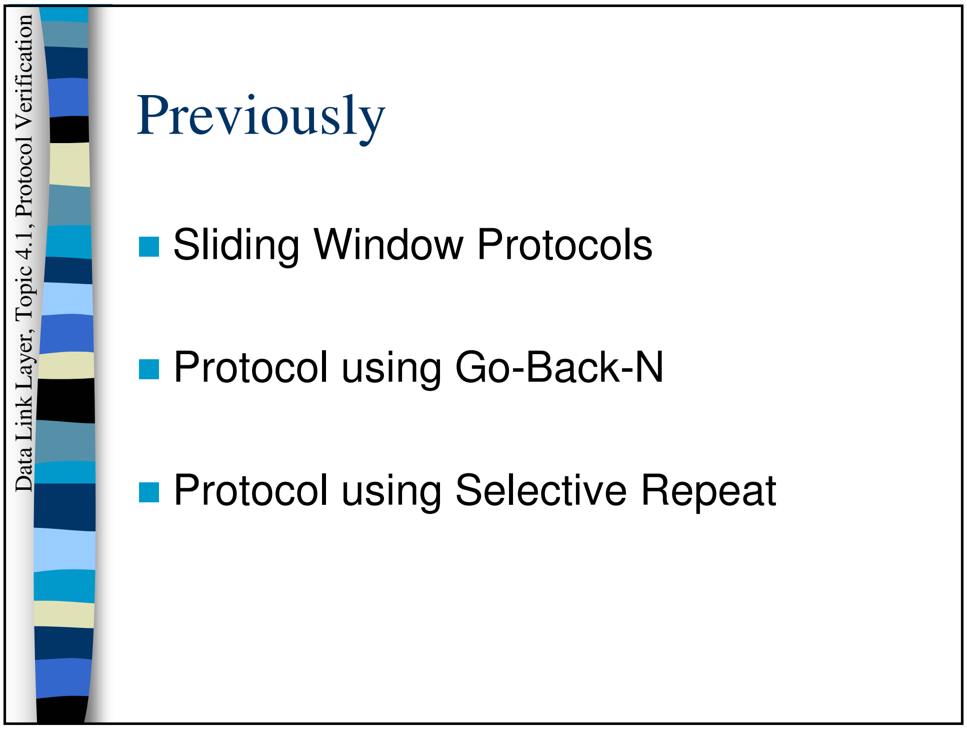
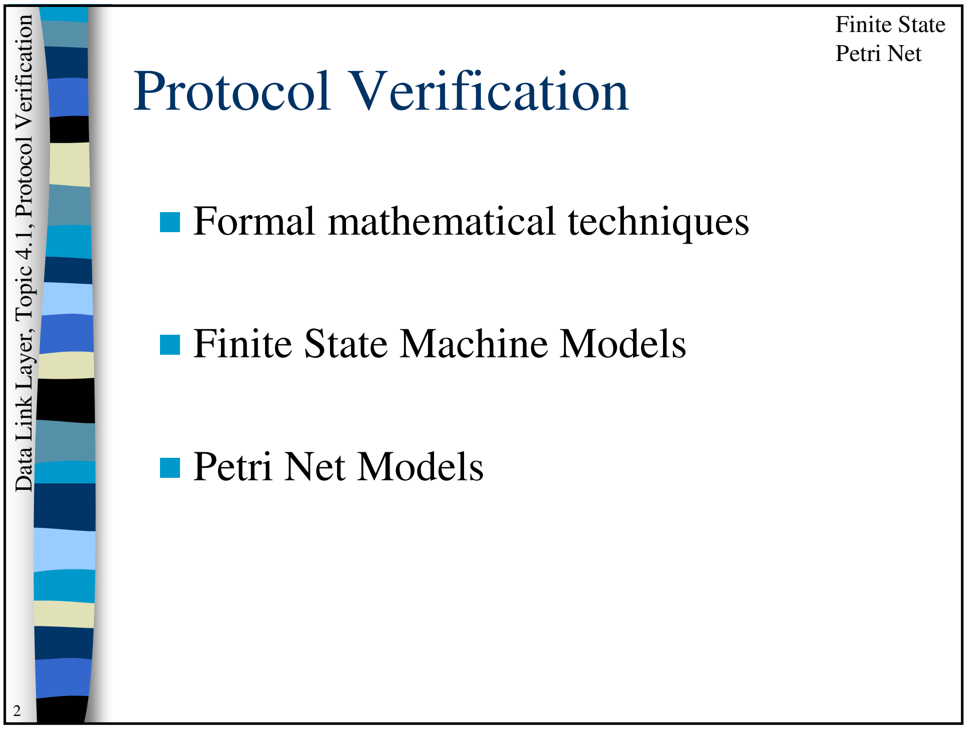


Previously

- Sliding Window Protocols
- Protocol using Go-Back-N
- Protocol using Selective Repeat



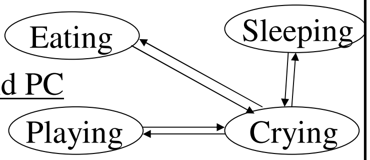
Finite State
Petri Net

Protocol Verification

- Formal mathematical techniques
- Finite State Machine Models
- Petri Net Models

Finite State Machines (1)

- Protocol Machines: i.e., sender or receiver
- State
  - Includes all variable values and PC
  - 2^n possible states,
  - **n** is the number of bits needed to represent all the variables,
  - This is very large so states are grouped together,
    - Generally chosen from those where the machine is waiting for some event, as all other states can be regarded as transient.



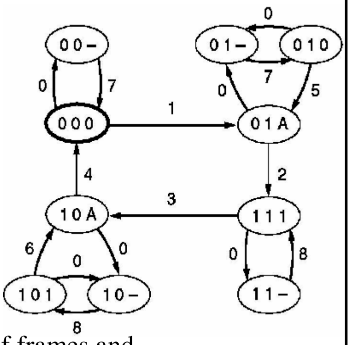
- Transitions: From each state there are 0 or more transitions to other states,
  - These are caused by e.g., receipt of a frame, loss of a frame,
- One state must be designated as initial state.

Finite State Machines (2)

- Given a full description of a FSM it should be possible to draw a graph
  - Nodes represent states,
  - Directed arcs represent transitions,
- Reachability Analysis allows to identify potential problems in FSM such as
  - Incompleteness: What to do if something happens in a particular state for which there is no transition,
  - Deadlock: Where there is no way out of a state or a group of states and where no progress can be made.

Finite State
Petri Net

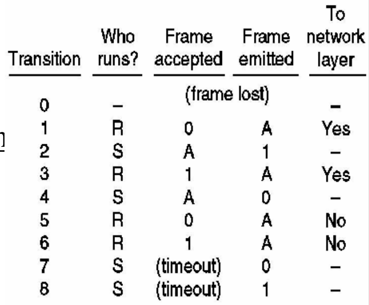
Example.



- Protocol for a Noisy Channel
- States
  - 3 variables
    - S The frame the sender has sent
    - R The frame the rcver is trying to rcv
    - C What is on the channel
- Initial State 000
- Normal transmission
  - 000 → 01A → 111 → 10A consists of frames and acknowledgements alternating on the channel (1,2,3,4 transitions)
- Lost data frame
  - 000 → 00– → 000 Frame 0 is lost ($\xrightarrow{0}$) and is retransmitted($\xrightarrow{7}$)
  - 111 → 11– → 111 Frame 1 is lost ($\xrightarrow{0}$) and is retransmitted($\xrightarrow{8}$)
- Lost acknowledgement
  - **01A → 01– → 010 → 01A** Causes transition 0 and retransmission (7) , followed by a repeated acknowledgement (5)
  - **10A → 10– → 101 → 10A** Causes transition 0 and retransmission (8) followed by a repeated acknowledgement (6)

Finite State
Petri Net

Checking for problems



Transition	Who runs?	Frame accepted	Frame emitted	To network layer
0	–	(frame lost)		–
1	R	0	A	Yes
2	S	A	1	–
3	R	1	A	Yes
4	S	A	0	–
5	R	0	A	No
6	R	1	A	No
7	S	(timeout)	0	–
8	S	(timeout)	1	–

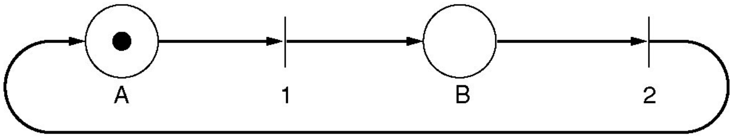
- Alternating frames
  - Never 11 without a 3 between
  - Or 33 without a 1 between
  - Check FSM…
- Deadlock
  - No way out of the subset,
  - No forward progress is being caused by any transitions in the subset,
  - Check FSM…

Data Link Layer, Topic 4.1, Protocol Verification

Finite State
Petri Net

Petri Net Models

- Places: Represent a state which (part of) the system may be in. [circle]
- Tokens: Indicates the place(s) that the system is currently in. [dot]
- Transitions: A possible change of place/state. [vertical bar]
  - Input Arcs: Arrows from the input places
  - Output Arcs: Arrows to the output places
  - Enabled: A transition is enabled when there are tokens in all of its input arcs,
  - Fire: A transition may fire at will once enabled
    - Effect: Removes tokens from all input places and places a token in each of its output places,
    - When? More than a single transition may be enabled at the same time and the choice of which to fire is indeterminate.

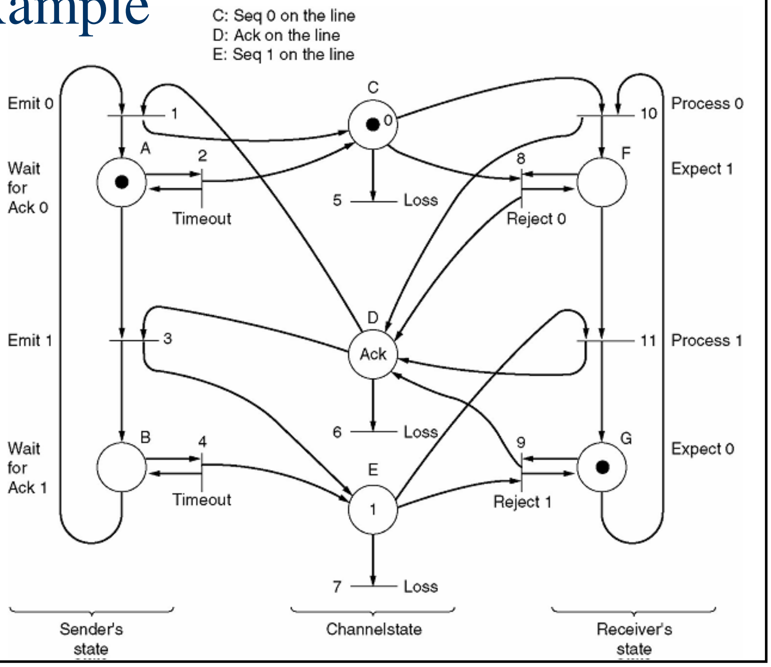


7

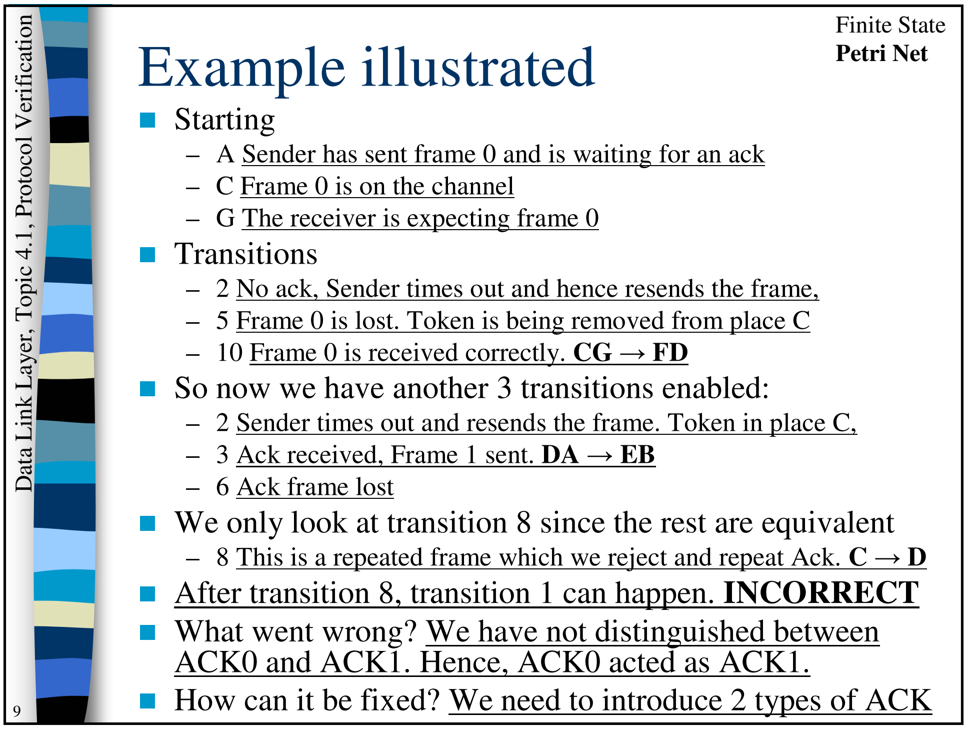
Data Link Layer, Topic 4.1, Protocol Verification

Finite State
Petri Net

Example



8



Finite State
Petri Net

Example illustrated

- Starting
  - A Sender has sent frame 0 and is waiting for an ack
  - C Frame 0 is on the channel
  - G The receiver is expecting frame 0
- Transitions
  - 2 No ack, Sender times out and hence resends the frame,
  - 5 Frame 0 is lost. Token is being removed from place C
  - 10 Frame 0 is received correctly. **CG → FD**
- So now we have another 3 transitions enabled:
  - 2 Sender times out and resends the frame. Token in place C,
  - 3 Ack received, Frame 1 sent. **DA → EB**
  - 6 Ack frame lost
- We only look at transition 8 since the rest are equivalent
  - 8 This is a repeated frame which we reject and repeat Ack. **C → D**
- After transition 8, transition 1 can happen. **INCORRECT**
- What went wrong? We have not distinguished between ACK0 and ACK1. Hence, ACK0 acted as ACK1.
- How can it be fixed? We need to introduce 2 types of ACK