
# Session 5D

## Link State Routing: OSPF

**Ref: Text book 1**

**Mouli Sankaran**

# Session 5D: Focus

- Link State Routing Algorithm – Introduction
  - LSP: Link State Packet
  - Reliable Flooding of LSP
  - Reasons for LSP Generation
- LSA Design Goals
  - LSA: Sequence Number
  - LSA: TTL
  - Dijkstra's Shortest Path Algorithm: Example
- OSPF – LSA Based
- Difference between DVA and LSA
- Comparison: RIPv1 Vs RIPv2 Vs OSPF

**Course page where the course materials will be posted as the course progresses:**

# Link State Routing Algorithm

# Link State Routing Algorithm

- Link-state routing Algorithm is the second major class of intra-domain routing protocol.
- The starting assumptions for link-state routing are rather similar to those for distance-vector routing.
- Each node is assumed to be capable of finding out the state of the link to its neighbors (up or down) and the cost of each link.
- The aim of this protocol is to provide each node with enough information to enable it to find the least-cost path to any destination.
- The basic idea behind link-state protocols is very simple:
- Every node knows how to reach its directly connected neighbors, and if we make sure that the totality of this knowledge is disseminated to every node.
  - Which means that each node is given all the information that every other node has generated about its directly connected neighbours

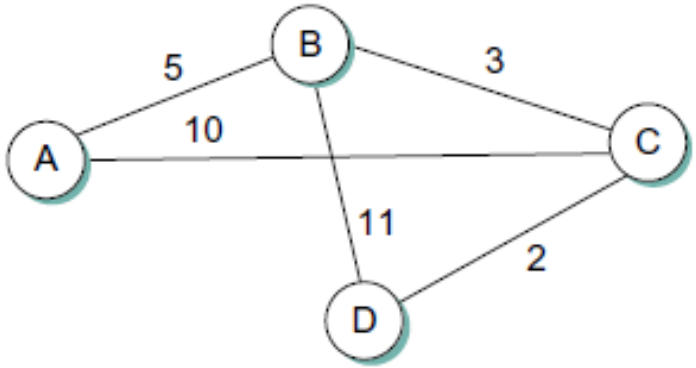
# Link State Routing: Path Finding

- Then every node will have enough knowledge of the network to build a complete map of the network.
- This is clearly a sufficient condition (although not a necessary one) for finding the shortest path to any point in the network.
- Thus, link state routing protocols rely on two mechanisms:

1. Reliable dissemination of link-state information
2. The calculation of routes from the sum of all the accumulated link-state knowledge

# Link State Advertisement (LSA) or LSP

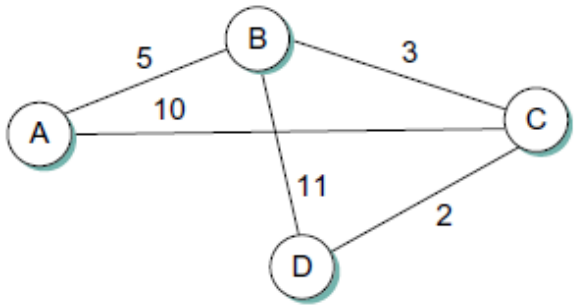
- LSA which is also called as **LSP** (**Link State Packet**) is the information that every router in the network shares with every other node or router in the network (routing domain) – **using reliable flooding (next slide)**




- LSA contains the following information: (For example **from D**)
- The ID of the node that created the LSP → **D**
- A list of directly connected neighbors of that node, with the cost of the link to each one **(Neighbour, cost, next hop) → (B, 11, B), (C, 2, C)**
- A sequence number → **A 64 bits wide number which starts with zero, never expected to wrap round**
- A **LSA Age** for this packet → **It is different from the TTL in IP packet**
  - It is **initialized to zero** by the originator.
  - It is **incremented** by the routers while passing it to others.
  - It is **incremented** based on the **time elapsed** while LSA is stored in the routers.
  - When it ages (1 hour), it is discarded. Originator needs to refresh it.

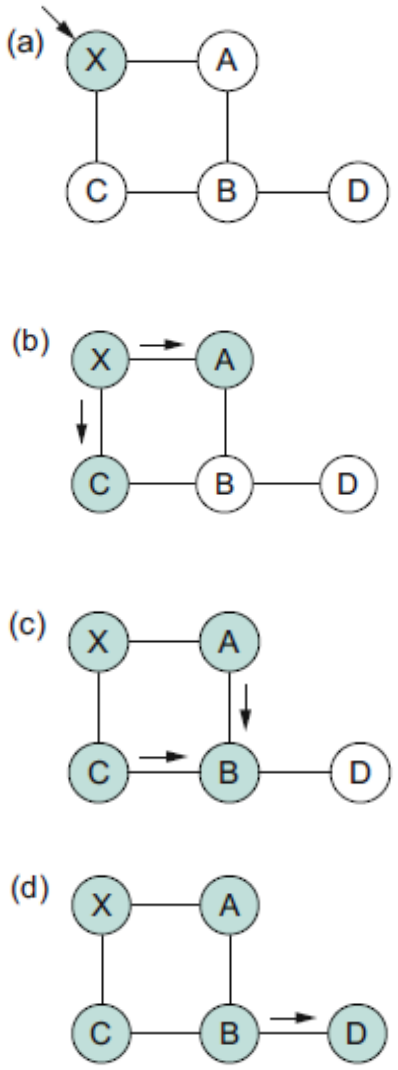
# Reliable Flooding of LSP

- The first stage of the LSA is sharing of LSP by every node participating in the LSA with every other node in the routing domain.
- Each node **keeps** the **latest** and the **current LSP** received from every other node. (based on **Seq #**)




- Flooding works in the following way.
- First, the transmission of LSPs between adjacent routers is made reliable using **acknowledgments** for every LSP message exchanged.
- However, several more steps are necessary to reliably flood an LSP to all the nodes (routers) in the network (routing domain).
- **Sequence number** is used to make sure that the receiving node always keeps the **latest LSP** of each node and discards the older one.
- This makes sure that the correct and the latest LSP of every node is available at every other node, for each of them to independently find the shortest path to every other node in the domain. **(distributed algorithm)**

# Reliable Flooding of LSP: Example




- It shows an LSP being flooded in a small network.
- Each node becomes shaded as it stores the new LSP.
- When an LSP arrives at node X, it sends it to its neighbors A and C, that is flooding the received LSP to all the other node it is connected to. (**Ref** the note below)
- A and C do not send it back to X, but they both send it on to B.
- Since B receives two identical copies of the LSP, it will accept whichever arrived first and ignore the second as a duplicate.
- B then passes the LSP onto D, which has no neighbors to flood it to, and the process is complete.
- Let us see when does a node send an LSP …

**Note**: Not all LSA are not accepted by a node. If a stale LSA is received it is discarded and not forwarded further (based on the Seq #).

# Generation of LSP: Reasons

- Just as in RIP, each node generates LSPs under two circumstances.

1. Either the expiry of a periodic timer (**1 hour**) or
2. A change in topology causes a node to generate a new LSP.

- However, the only topology-based reason for a node to generate an LSP is, if one of its directly connected links or immediate neighbors has gone down.
- The failure of a link can be detected in some cases by the link-layer protocol.
- The demise of a neighbor or loss of connectivity to that neighbor can be detected using periodic “hello” packets.
- Each node sends “hello” to its immediate neighbors at defined intervals.
- If a sufficiently long time passes without receipt of a “hello” from a neighbor, the links to that neighbor will be declared down and a new LSP will be generated to reflect this fact, by the node which detects it.

# LSA: Design Goals

- One of the important design goals of a link-state protocol's flooding mechanism is that the newest information must be flooded to all nodes as quickly as possible, while old information must be removed from the network and not allowed to circulate.
- In addition, it is clearly desirable to minimize the total amount of routing traffic that is sent around the network;
- Because this is just overhead from the perspective of those who actually use the network for their applications.
- One easy way to reduce overhead is to avoid generating LSPs unless absolutely necessary.
- This can be done by using very long timers—often on the order of hours—for the periodic generation of LSPs.
- Since flooding is reliable, having long timers is acceptable.

# LSA: Design Goals …. contd.

- Given that the flooding protocol is truly reliable when topology changes, it is safe to assume that messages saying "nothing has changed" do not need to be sent very often.
- Remember, the periodic "hello" messages are only sent to the neighbours and these "hello' messages are not flooded by the receiving nodes. What is flooded is the LSP.
- Sequence number in LSP helps in finding and replacing the stale LSP stored in a node and it makes sure that all the nodes have the latest information on the topology of the entire routing domain.
- Unlike most sequence numbers used in protocols, these sequence numbers are not expected to wrap, so the field needs to be quite large (say, 64 bits).

# LSA Age: (equivalent to TTL)

- LSPs also carry a time to live field element.
- This is used to ensure that old link state information is eventually removed from the network.
- A node **increments** the **LSA Age** field of a newly received LSP before flooding it to its neighbors.
- Each node also "ages" the LSP periodically while it is stored in the node.
- That is, each node increments the LSA Age of LSPs of every node it has, which is set back to received value when the node receives a new LSP
- When the LSA Age reaches **Max Age** of any node, the node re-floods that it with a value of Max Age, which is interpreted by all the nodes in the network as a signal to delete that LSP that they have with them.
- This action makes sure that any node which is dead is removed from all the nodes, by removing the LSP belonging to the dead node.

# LSA: Sequence Number

- If a node goes down and then comes back up, it starts with a sequence number of 0.
- If the node was down for a long time, all the old LSPs for that node would have timed out (because of LSA Age field in LSP);
- Otherwise, this node will eventually receive a copy of its own LSP with a higher sequence number (which it had sent or flooded before going down)
- Now, it can use its own previous old sequence number by incrementing it by one and use as its own sequence number.
- By this, other nodes in the network won't even know that this node went down and came back to life ☺
- And, this also ensures that its new LSP replaces any of its old LSPs left over on other nodes before the node went down.

# Dijkstra's Shortest Path algorithm (OSPF)

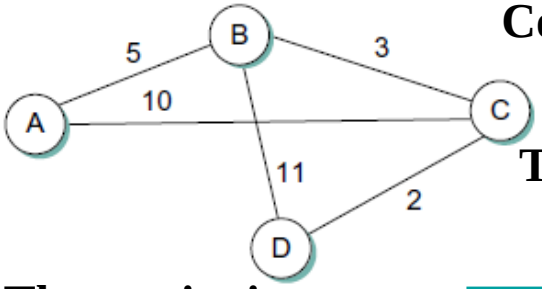
# What is the next step?

- We have made sure that every node, part of the LSA routing domain has the following information with them:

1. The latest and valid LSPs of every other node which are currently alive.
2. Each node has the most reliable information about every other node which it has received from those nodes themselves, giving information about their neighbours and the cost to reach each of them from it.

- Now the next step is how every node uses this global network topology information to independently find the shortest path to all the nodes in the network routing domain.
- Each node runs **Dijkstra's Shortest Path algorithm** to find the shortest path to every other node!!!

# Dijkstra's Shortest Path Algorithm: Example
## (Algorithm is running on Node D)




**Confirmed List:** It has the shortest path from the current node to other nodes which are confirmed and finalized.

**Tentative List:** It has the nodes for which the shortest paths are yet to be finalized from the node which is running this algorithm.

**The entries in the list are**:
(Neighbor, Cost, NextHop),

The algorithm ends when all the nodes are added to the confirmed list.

Each node runs this algorithm and finds out shortest paths to all other nodes.

**Table 3.14 Steps for Building Routing Table for Node D**

| Step | Confirmed | Tentative | Comments |
|---|---|---|---|
| 1 | (D,0,–) | | Since D is the only new member of the confirmed list, look at its LSP. |
| 2 | (D,0,–) | (B,11,B) (C,2,C) | D's LSP says we can reach B through B at cost 11, which is better than anything else on either list, so put it on Tentative list; same for C. |
| 3 | (D,0,–) (C,2,C) | (B,11,B) | Put lowest-cost member of Tentative (C) onto Confirmed list. Next, examine LSP of newly confirmed member (C). |
| 4 | (D,0,–) (C,2,C) | (B,5,C) (A,12,C) | Cost to reach B through C is 5, so replace (B,11,B). C's LSP tells us that we can reach A at cost 12. |
| 5 | (D,0,–) (C,2,C) (B,5,C) | (A,12,C) | Move lowest-cost member of Tentative (B) to Confirmed, then look at its LSP. |
| 6 | (D,0,–) (C,2,C) (B,5,C) | (A,10,C) | Since we can reach A at cost 5 through B, replace the Tentative entry. |
| 7 | (D,0,–) (C,2,C) (B,5,C) (A,10,C) | | Move lowest-cost member of Tentative (A) to Confirmed, and we are all done. |

# Dijkstra's Shortest Path Algorithm: Explained

1. Initialize the Confirmed list with an entry for myself; this entry has a cost of 0.
2. For the node just added to the Confirmed list in the previous step, call it node Next and select its LSP.
3. For each neighbor (Neighbor) of Next, calculate the cost (Cost) to reach this Neighbor as the sum of the cost from myself to Next and from Next to Neighbor.
   (a) If Neighbor is currently on neither the Confirmed nor the Tentative list, then add (Neighbor, Cost, NextHop) to the Tentative list, where NextHop is the direction I go to reach Next.
   (b) If Neighbor is currently on the Tentative list, and the Cost is less than the currently listed cost for Neighbor, then replace the current entry with (Neighbor, Cost, NextHop), where NextHop is the direction I go to reach Next.
4. If the Tentative list is empty, stop. Otherwise, pick the entry from the Tentative list with the lowest cost, move it to the Confirmed list, and return to step 2.

# OSPF: Open Shortest Path First

- **Open Shortest Path First** (**OSPF**) is an adaptive routing protocol for Internet Protocol (IP) networks. It uses a link state routing algorithm and falls into the group of interior routing protocols, operating within a single **autonomous system** (AS).
- OSPF is perhaps the most widely-used interior gateway protocol (IGP) in large enterprise networks. **IS-IS**, another link-state dynamic routing protocol, is more common in large service provider networks.

# Difference Between DVA and LSA

The difference between the distance-vector and link-state algorithms can be summarized as follows. In distance-vector, each node talks only to its directly connected neighbors, but it tells them everything it has learned (i.e., distance to all nodes). In link-state, each node talks to all other nodes, but it tells them only what it knows for sure (i.e., only the state of its directly connected links).

# RIPv1 Vs RIPv2 Vs OSPF

| Features | RIP | | OSPF |
|---|---|---|---|
| | Version 1 | Version 2 | |
| Algorithm | Bellman-Ford | | Dijkstra |
| Path Selection | Hop based | | Shortest Path |
| Routing | Classful | Classless | Classless |
| Transmission | Broadcast | Multicast | Multicast |
| Administrative Distance | 120 | | *110* |
| Hop Count Limitation | 15 | | No Limitation |
| Authentication | No | MD5 | MD5 |
| Protocol | UDP | | IP |
| Convergence Time | RIP>OSPF | | |

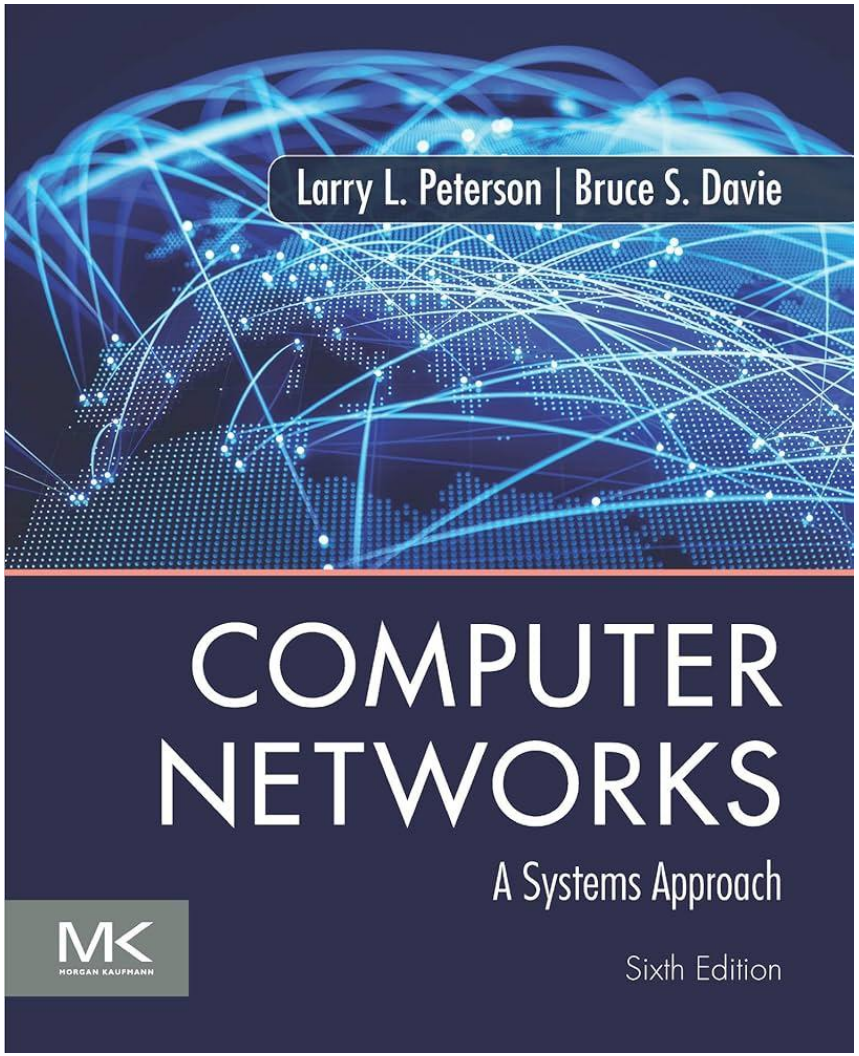
**Administrative distance (AD) or route preference**: is a number of arbitrary unit assigned to dynamic routes, static routes and directly-connected routes. The value is used by vendor-specific routers to rank routes from most preferred (low administrative distance value) to least preferred (high administrative distance value)

# Session 5D: Summary

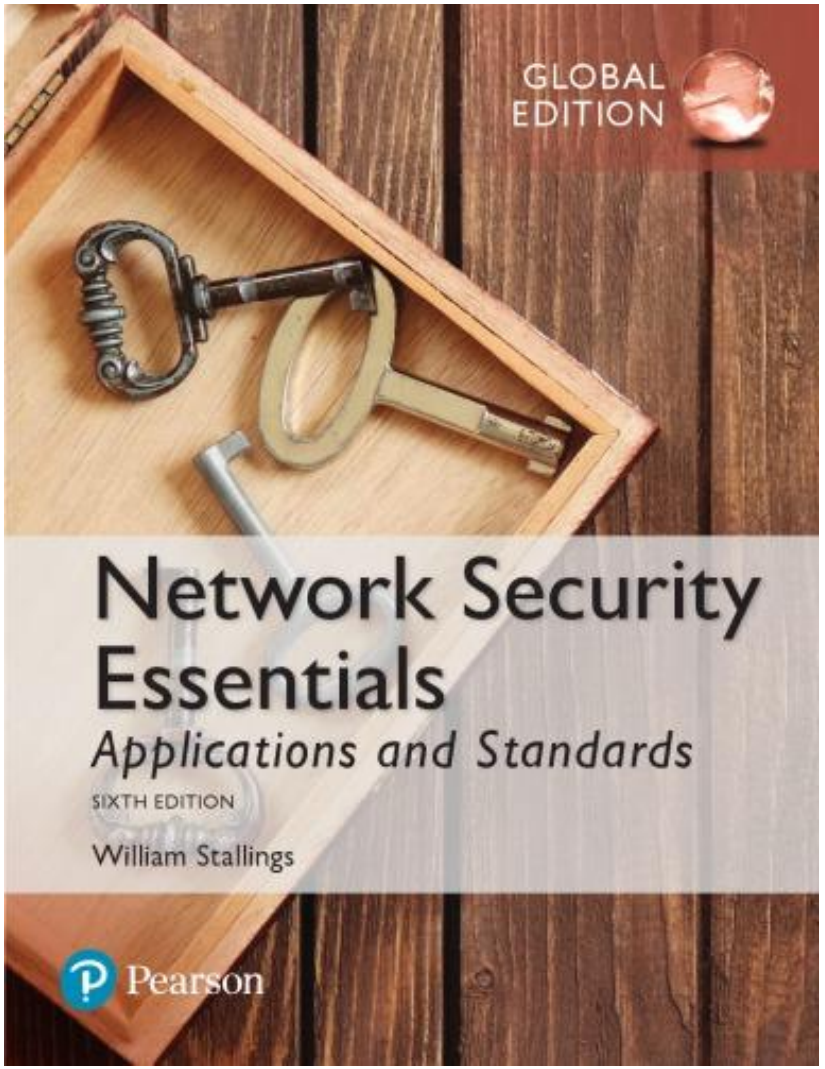
- Link State Routing Algorithm – Introduction
  - LSP: Link State Packet
  - Reliable Flooding of LSP
  - Reasons for LSP Generation
- LSA Design Goals
  - LSA: Sequence Number
  - LSA: TTL
  - Dijkstra's Shortest Path Algorithm: Example
- OSPF – LSA Based
- Difference between DVA and LSA
- Comparison: RIPv1 Vs RIPv2 Vs OSPF

# Textbooks

**Textbook 1**

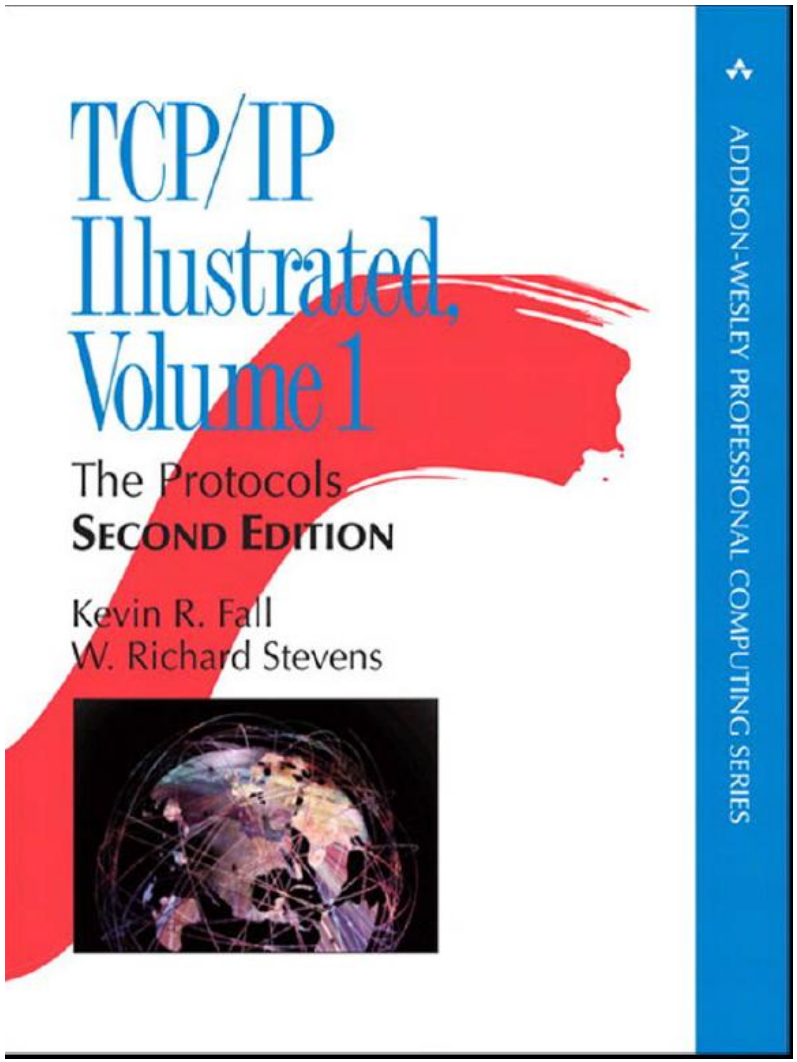



**Textbook 2**



# References

**Ref 1**




**Ref 2**

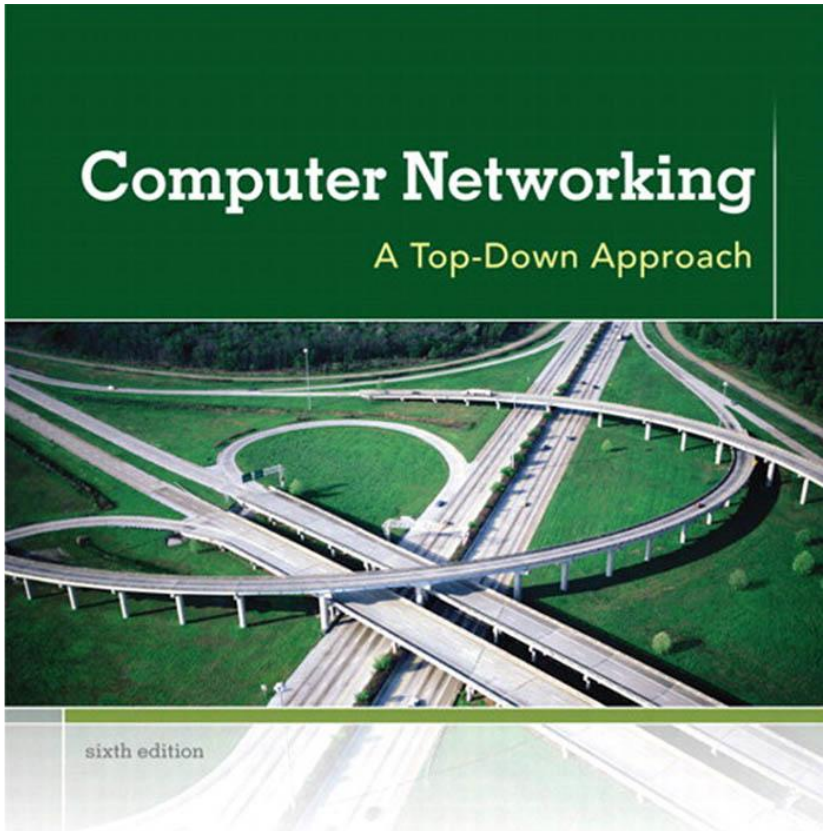
TCP Congestion Control: A Systems Approach


TCP Congestion Control: A Systems Approach

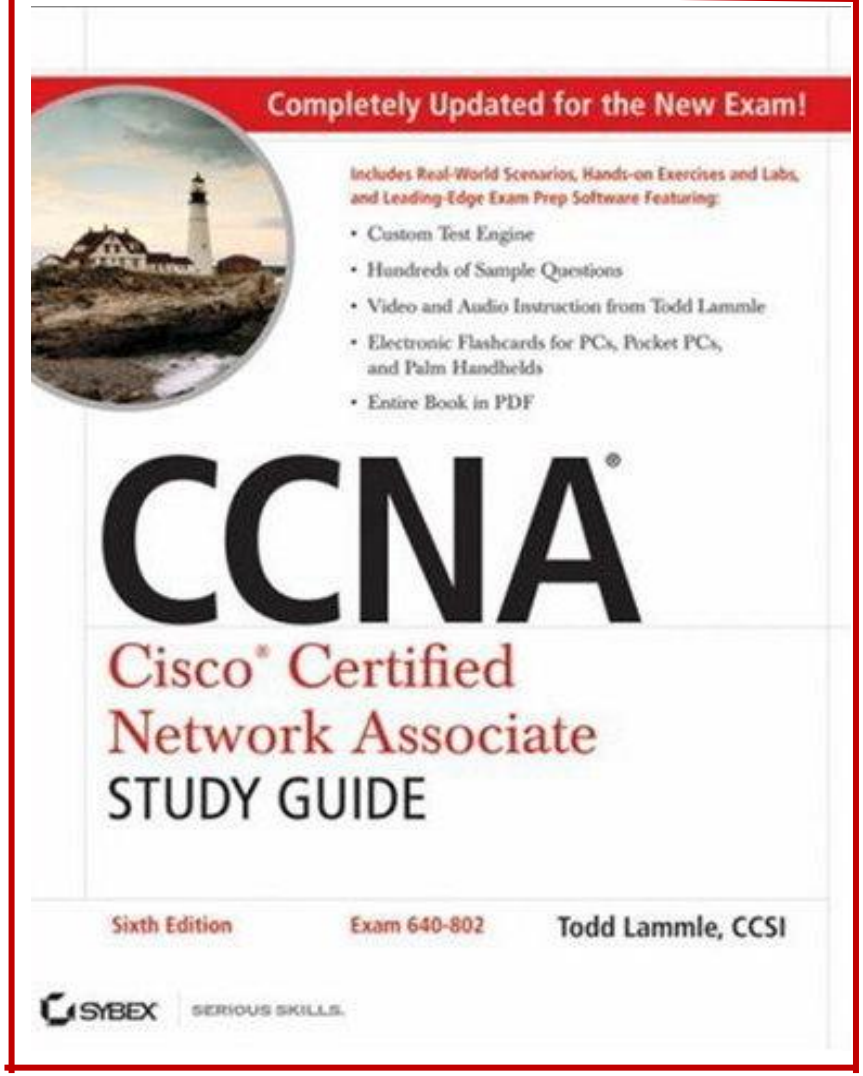
Peterson, Brakmo, and Davie

# References

**Ref 3**




**Ref 4**



# References

**Ref 5**