Session 3D

TCP: Nagle's Algorithm

Mouli Sankaran

Session 3D: Focus

- Silly Window Syndrome
  - Quiz 2: Generation of ACK
- Nagle's Algorithm
- TCP_NOREPLY option

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

Silly Window Syndrome

Meaning of Silly: Having or showing a lack of common sense or judgement; absurd and foolish.

Meaning of Syndrome: A group of symptoms which consistently occur together, or a condition characterized by a set of associated symptoms.

Quiz 1: Generation of ACKs

- Choose **all the options** related to the **condition(s)** on which a **receiver generates an ACK**.

A. When a data is read by the application on the receiving end and it is removed from the Rx Buf by the TCP/IP stack

B. When the data is received by the TCP/IP stack and it is buffered into the Rx Buf, but it does not wait for the application on its end to read the data from the Rx Buf.

C. The receiver may combine the sending of ACK along with its own data, if there is some data ready to be sent.

D. The receiver sends one combined ACK after all the data in its Rx Buf gets emptied, i.e., the application on its end reads all of them.

ANS: B and C

Note: Rx application consuming the data from Rx Buf influences the Window size not the ACK. Window size element is sent along with the ACK segment.

Note: Remember the ACK is exchanged at the TCP peer level, it does not have to wait for the application to consume the data. In fact if it does, it does not make use of the data buffering at the TCP level and also slows down the sender by not allowing to empty its own Tx buf.

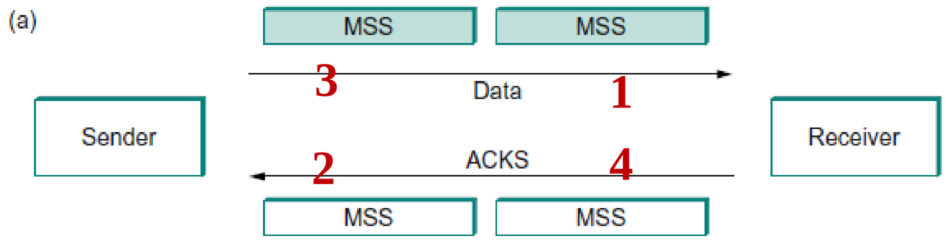
1. Silly Window Syndrome

- Though MSSs are agreed upon between the peers, we can't just ignore flow control, which plays an obvious role in throttling the sender.
- If the sender has MSS bytes of data to send and the window is open at least that much, then the sender transmits a full segment.
- Assume that the sender is accumulating bytes to send, but the window is currently closed.
- Now suppose an **ACK arrives** along with **the window size** that effectively **opens** the **window** enough for the sender to transmit, say, **MSS/2 bytes**. **What would have triggered this MSS/2 window size?**

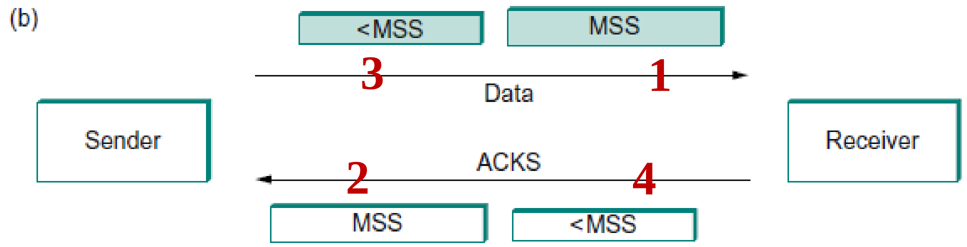
  ANS: The application at the receiver has read MSS/2 data from the Rx buffer.
- Should the sender transmit a half-full segment or wait for the window to open to a full MSS? **Why should it wait for full MSS?**

  ANS: If full MSS data is sent in a single TCP segment, it is more efficient.
- Let us see an example …

2. Silly Window Syndrome: Example



- As long as the sender sends MSS-sized segments and the receiver ACKs one MSS at a time and opens window size also by MSS, the system works smoothly.



- As soon as the **sender sends data which is less than one MSS**, or the receiver ACKs less than one MSS, a small "**container**" or "**segment**" enters the system and continues to circulate.
  - Sender sends less than MSS though window size is MSS, because it has only less data in TxBuf which is available to be sent
- Which is not good from the network utilization perspective since smaller amount of data gets transmitted for a fixed overhead of 40 bytes.

3. Silly Window Syndrome

- It turns out that the strategy that **aggressively** takes advantage of any available window, leads to a situation now known as the **silly window syndrome**.
  - Which means that the sender deciding to send whatever the window size available, without waiting for it to be equal to MSS or more.
- The strategy here refers to the sender reacting to the availability of window sizes, which are smaller than MSS, by sending immediately a segment with a smaller size.
- Let us see **Nagle's algorithm** which addresses the problem with the Silly Window Syndrome.

Nagle’s Algorithm

Note: Small segments are called **tinygrams**. Nagle’s algorithm avoids tinygrams on the network thus improving the network performance.

Nagle's Algorithm: Background

- To avoid Silly window syndrome, if there is data to send but the window is open less than MSS, then TCP may want to wait for some amount of time before sending the available data, but the question is how long?
- If the TCP waits too long, then it hurts interactive applications like Telnet.
- If TCP doesn't wait long enough, then it risks sending a bunch of tiny packets and falling into the silly window syndrome.
- The answer is to introduce a timer and to transmit when the timer expires.
- While we could use a **clock-based timer** -for example, one that fires every 100 ms— **Nagle** introduced an elegant **self-clocking solution**
  - Fixed clock-based timer is not a preferred solution because it does not adapt to the technology advances in the networking and processors.
  - Fixed delay may hurt the performance of recent advanced applications.

Nagle's Algorithm: Background contd …

- The idea of this algorithm is that **as long as TCP** has some **data in flight**, from Tx to Rx, the sender will eventually receive an ACK.
- This ACK can be treated like a timer firing, triggering the transmission of more data.
- But, why do we have to check if there is an ACK expected from the other end or not?
- You need to understand that the instance of TCP/IP stack created per connection (could be a process or a module) gets invoked based on any events getting triggered either from the top (application) or from the bottom (IP layer)
  1. **From the top**: When the application has something to transmit or it has consumed some data in the Rx buf causing the Window Size to increase.
  2. **From the bottom**: When the IP layer receives a TCP segment (may be with or without data) from the peer.
- **Nagle's algorithm** provides a simple, unified rule for deciding when to transmit: **Let us see it next …**

Note: Without data could be an ACK coming from the peer without its own data in the other direction.

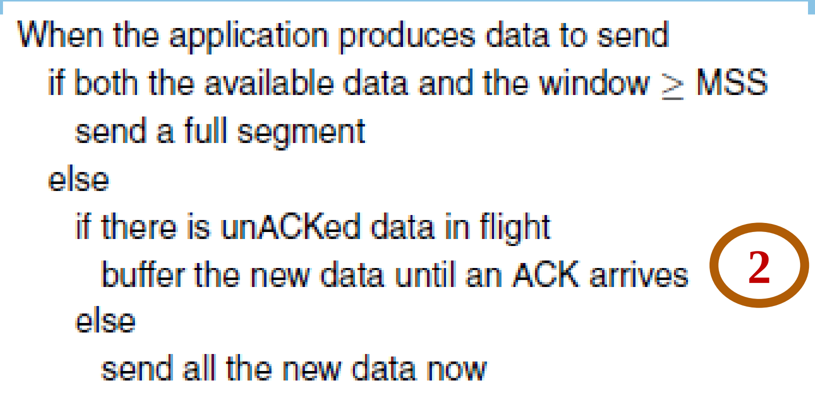
Nagle's Algorithm (1)

```
When the application produces data to send
  if both the available data and the window ≥ MSS
    send a full segment  1
  else
    if there is unACKed data in flight
      buffer the new data until an ACK arrives
    else
      send all the new data now
```

1. Full data of **max size MSS** can be sent only when **both** the following **two conditions** are **satisfied:**
   a) There is **sufficient data** (≥ **MSS**) in the Tx buf which is greater or equal to MSS.
   b) The receiver's **window size** (≥ **MSS**) allows full data to be sent.

- It's **always OK** to **send** a **full segment (MSS)** if **both the above conditions** are **true**.

Nagle's Algorithm (2)

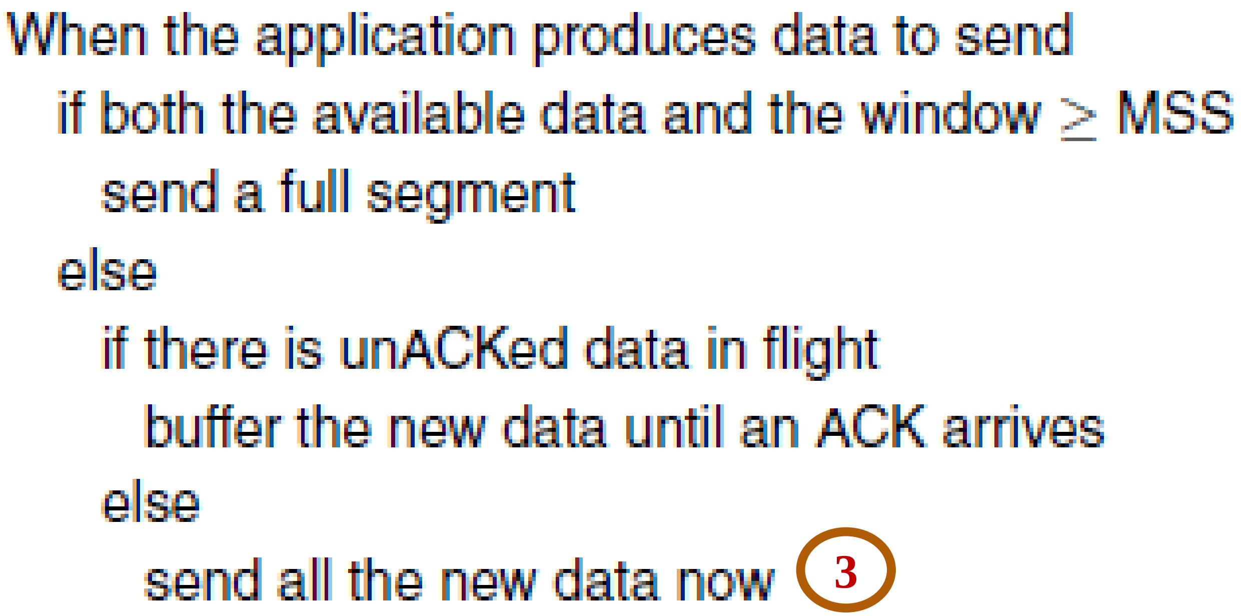


```
When the application produces data to send
  if both the available data and the window ≥ MSS
    send a full segment
  else
    if there is unACKed data in flight
      buffer the new data until an ACK arrives      2
    else
      send all the new data now
```

Note: Open-ended means having no predetermined limit or boundary.

2. It's all right to wait for the ACK from the other end if there are currently segments in transit, instead of initiating data transfer immediately which is smaller than MSS.
   - The reason being that an event from the other end is **not open-ended**.
   - If there is a data in transit, the peer is likely to generate an ACK on receiving the data, even if it does not have its own data to send from its end, so the sender can afford to wait for some time. During wait time the window size could also increase.
   - If there is no data in transit, there is no guarantee that some data would come from the application at the top triggering an event too, so it is not advisable to wait. because it could be open-ended wait, impacting interactive application's requirements.

Nagle's Algorithm (3)



```
When the application produces data to send
  if both the available data and the window ≥ MSS
    send a full segment
  else
    if there is unACKed data in flight
      buffer the new data until an ACK arrives
    else
      send all the new data now
```

3

3. So, it's also all right to immediately send a small amount of data, even if it is smaller than MSS, if there are currently no segments in transit.

TCP_NODELAY option

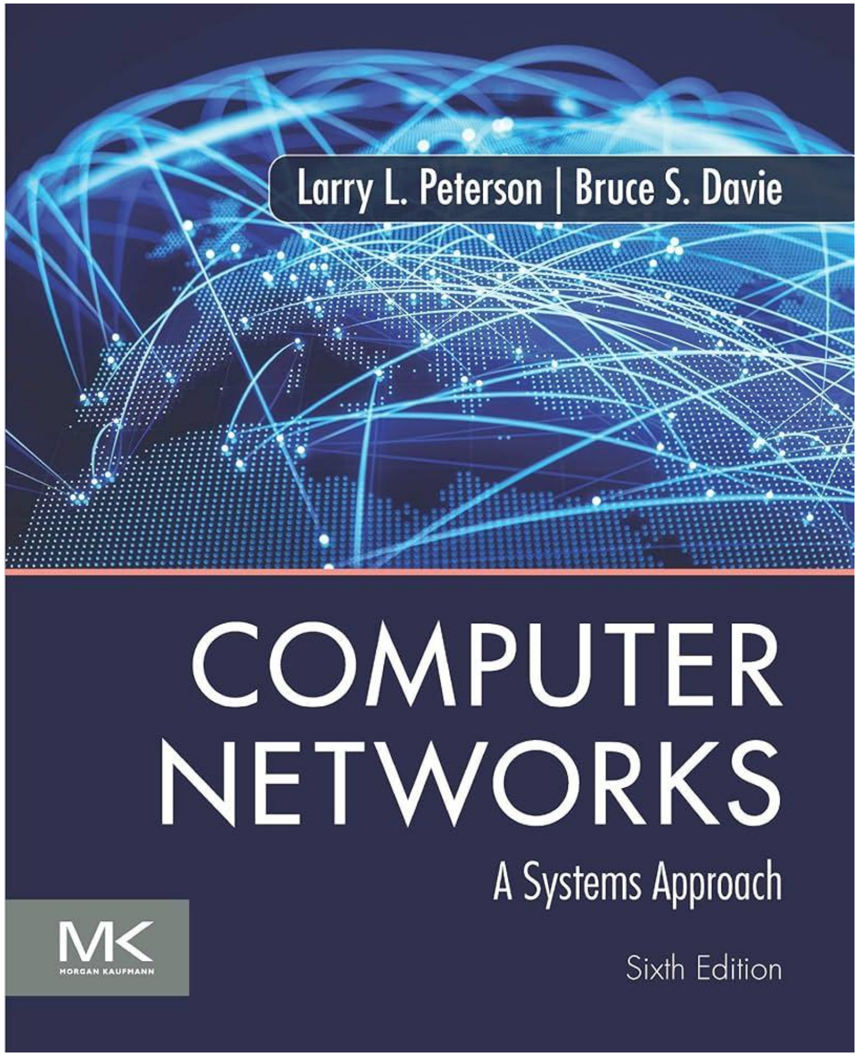
- An interactive application like Telnet that continually writes one byte at a time will send data at a rate of one segment per RTT (Round Trip Time).
- Some segments will contain a single byte, while others will contain as many bytes as the user was able to type in one round-trip time.
- Because some applications (**Telnet**) cannot afford such a delay for each write it does to a TCP connection, the socket interface allows the application to **turn off Nagel's algorithm** by **setting** the **TCP_NODELAY** option.
- Setting this option means that data is transmitted as soon as possible.
- RTT is chosen here because that is the maximum delay that would be experienced to get a response for the commands typed on one end. RTT is much lower than delay introduced by Nagel's algorithm.

Session 3D: Summary

- Silly Window Syndrome
  - Quiz 2: Generation of ACK
- Nagle’s Algorithm
- TCP_NOREPLY option

Textbooks

Textbook 1



Larry L. Peterson | Bruce S. Davie

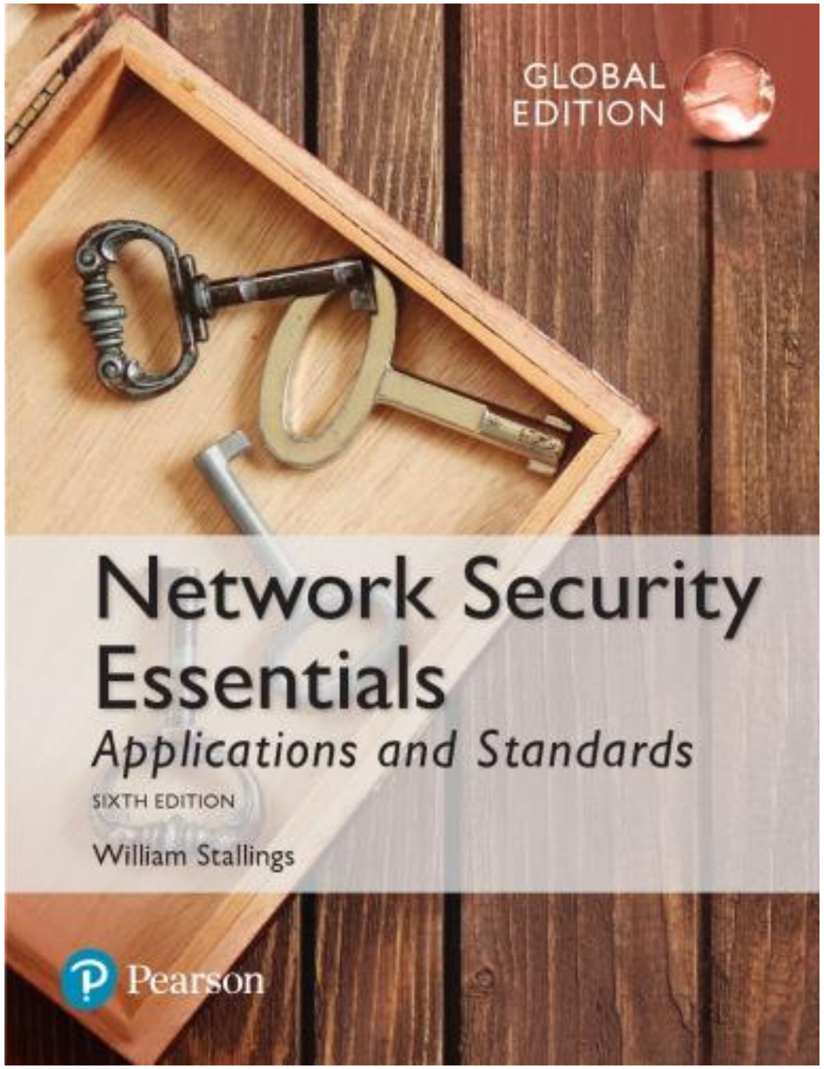
COMPUTER NETWORKS

A Systems Approach

MK MORGAN KAUFMANN

Sixth Edition

Textbook 2



GLOBAL EDITION

Network Security Essentials

Applications and Standards

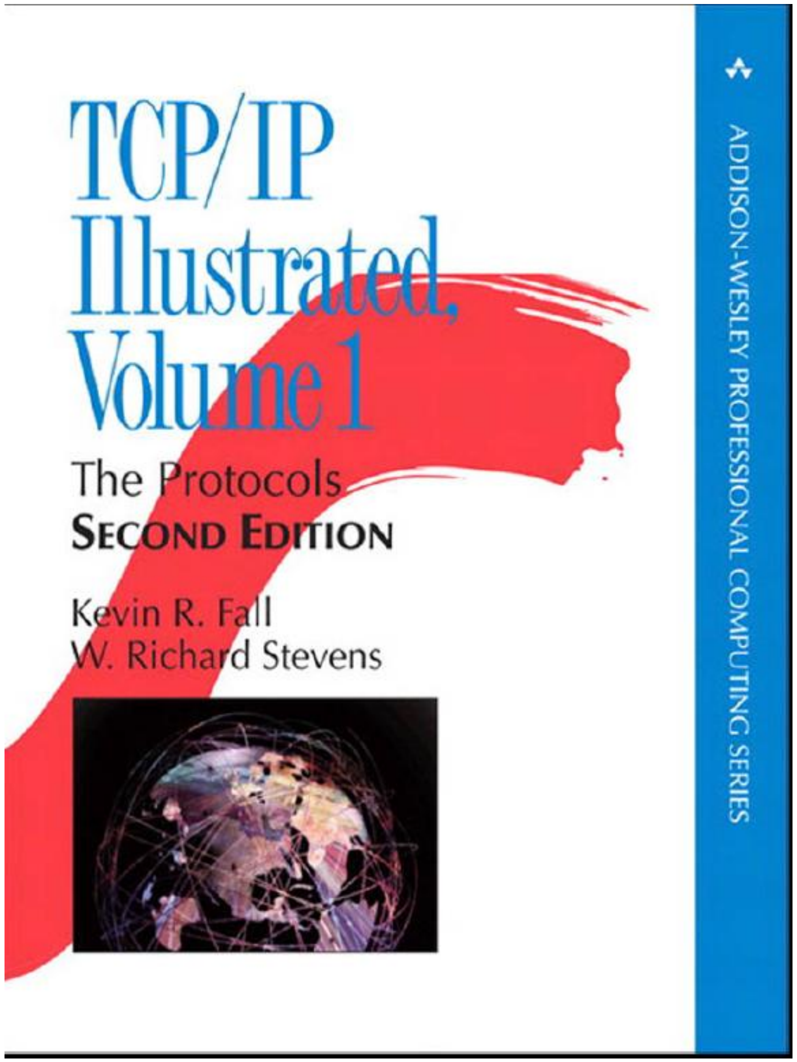
SIXTH EDITION

William Stallings

Pearson

References

Ref 1



TCP/IP Illustrated, Volume 1

The Protocols

SECOND EDITION

Kevin R. Fall
W. Richard Stevens

ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

Ref 2

[TCP Congestion Control: A Systems Approach]

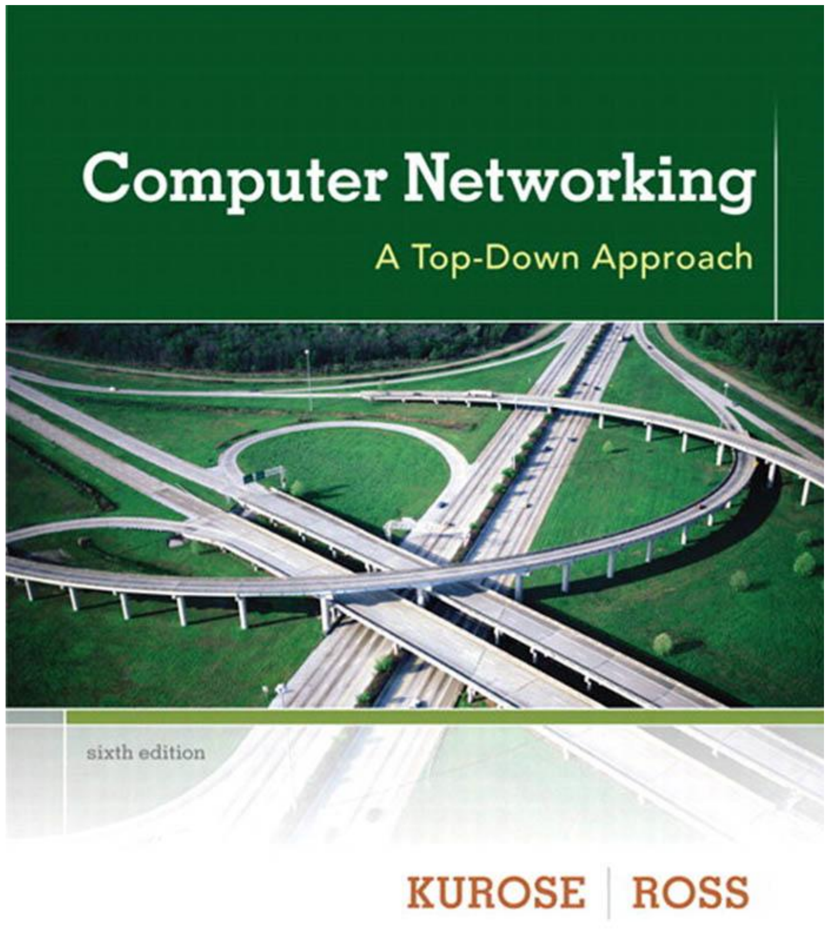
HTTPS://SYSTEMSAPPROACH.ORG

TCP Congestion Control: A Systems Approach

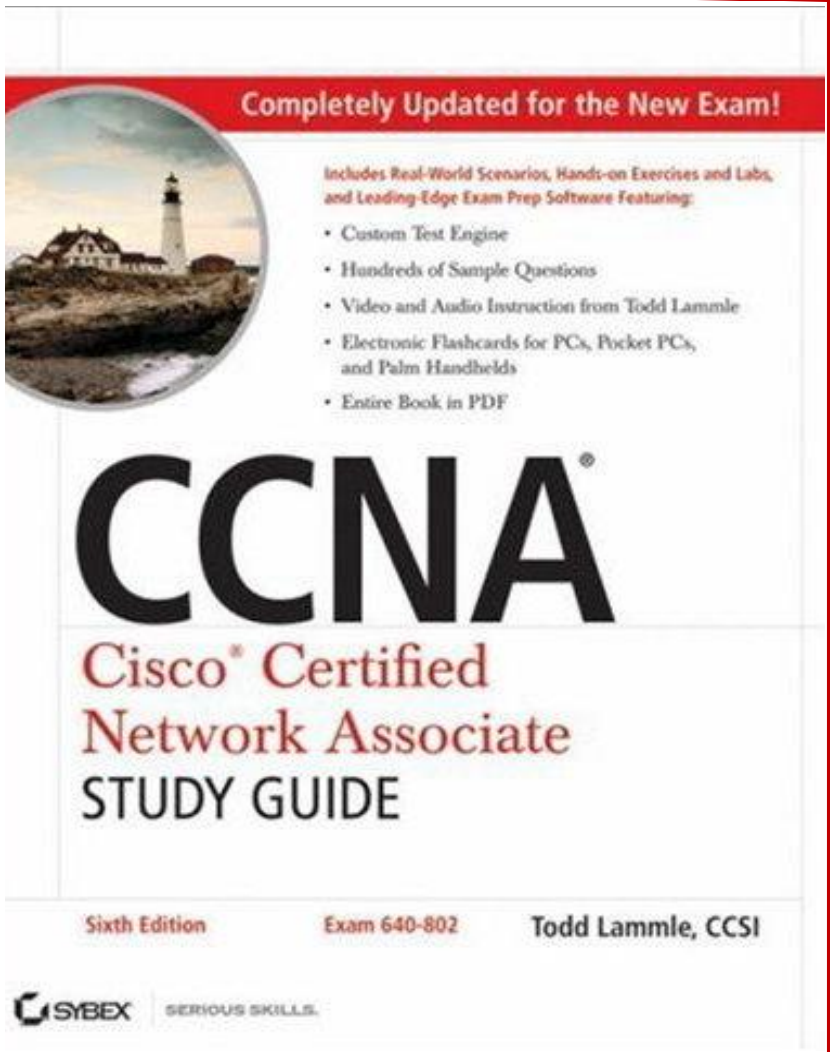
Peterson, Brakmo, and Davie

References

Ref 3



Ref 4


Completely Updated for the New Exam!
Includes Real-World Scenarios, Hands-on Exercises and Labs, and Leading-Edge Exam Prep Software Featuring:
• Custom Test Engine
• Hundreds of Sample Questions
• Video and Audio Instruction from Todd Lammle
• Electronic Flashcards for PCs, Pocket PCs, and Palm Handhelds
• Entire Book in PDF
CCNA®
Cisco® Certified Network Associate
STUDY GUIDE
Sixth Edition
Exam 640-802
Todd Lammle, CCSI
SYBEX
SERIOUS SKILLS.