A proposed modification to TCP, the so-called **selective acknowledgment** [RFC 2018], allows a TCP receiver to acknowledge out-of-order segments selectively rather than just cumulatively acknowledging the last correctly received, in-order segment. When combined with selective retransmission—skipping the retransmission of segments that have already been selectively acknowledged by the receiver—TCP looks a lot like our generic SR protocol. Thus, TCP's error-recovery mechanism is probably best categorized as a hybrid of GBN and SR protocols.

3.5.5 Flow Control

Recall that the hosts on each side of a TCP connection set aside a receive buffer for the connection. When the TCP connection receives bytes that are correct and in sequence, it places the data in the receive buffer. The associated application process will read data from this buffer, but not necessarily at the instant the data arrives. Indeed, the receiving application may be busy with some other task and may not even attempt to read the data until long after it has arrived. If the application is relatively slow at reading the data, the sender can very easily overflow the connection's receive buffer by sending too much data too quickly.

TCP provides a **flow-control service** to its applications to eliminate the possibility of the sender overflowing the receiver's buffer. Flow control is thus a speed-matching service—matching the rate at which the sender is sending against the rate at which the receiving application is reading. As noted earlier, a TCP sender can also be throttled due to congestion within the IP network; this form of sender control is referred to as **congestion control**, a topic we will explore in detail in Sections 3.6 and 3.7. Even though the actions taken by flow and congestion control are similar (the throttling of the sender), they are obviously taken for very different reasons. Unfortunately, many authors use the terms interchangeably, and the savvy reader would be wise to distinguish between them. Let's now discuss how TCP provides its flow-control service. In order to see the forest for the trees, we suppose throughout this section that the TCP implementation is such that the TCP receiver discards out-of-order segments.

TCP provides flow control by having the *sender* maintain a variable called the **receive window**. Informally, the receive window is used to give the sender an idea of how much free buffer space is available at the receiver. Because TCP is full-duplex, the sender at each side of the connection maintains a distinct receive window. Let's investigate the receive window in the context of a file transfer. Suppose that Host A is sending a large file to Host B over a TCP connection. Host B allocates a receive buffer to this connection; denote its size by `RcvBuffer`. From time to time, the application process in Host B reads from the buffer. Define the following variables:

- `LastByteRead`: the number of the last byte in the data stream read from the buffer by the application process in B
- `LastByteRcvd`: the number of the last byte in the data stream that has arrived from the network and has been placed in the receive buffer at B

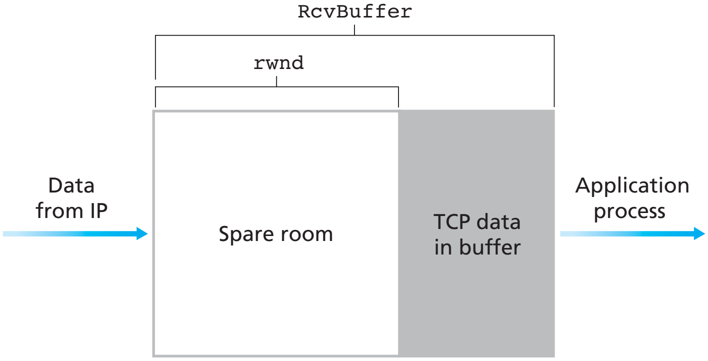

Figure 3.38 ♦ The receive window (`rwnd`) and the receive buffer (`RcvBuffer`)

Because TCP is not permitted to overflow the allocated buffer, we must have

`LastByteRcvd – LastByteRead ≤ RcvBuffer`

The receive window, denoted `rwnd` is set to the amount of spare room in the buffer:

`rwnd = RcvBuffer – [LastByteRcvd – LastByteRead]`

Because the spare room changes with time, `rwnd` is dynamic. The variable `rwnd` is illustrated in Figure 3.38.

How does the connection use the variable `rwnd` to provide the flow-control service? Host B tells Host A how much spare room it has in the connection buffer by placing its current value of `rwnd` in the receive window field of every segment it sends to A. Initially, Host B sets `rwnd` = `RcvBuffer`. Note that to pull this off, Host B must keep track of several connection-specific variables.

Host A in turn keeps track of two variables, `LastByteSent` and `LastByteAcked`, which have obvious meanings. Note that the difference between these two variables, `LastByteSent` – `LastByteAcked`, is the amount of unacknowledged data that A has sent into the connection. By keeping the amount of unacknowledged data less than the value of `rwnd`, Host A is assured that it is not overflowing the receive buffer at Host B. Thus, Host A makes sure throughout the connection's life that

`LastByteSent – LastByteAcked ≤ rwnd`