Higher Layer Protocols: UDP, TCP, ATM, MPLS

Eytan Modiano
Massachusetts Institute of Technology

The TCP/IP Protocol Suite

- **Transmission Control Protocol / Internet Protocol**
- **Developed by DARPA to connect Universities and Research Labs**

Four Layer model

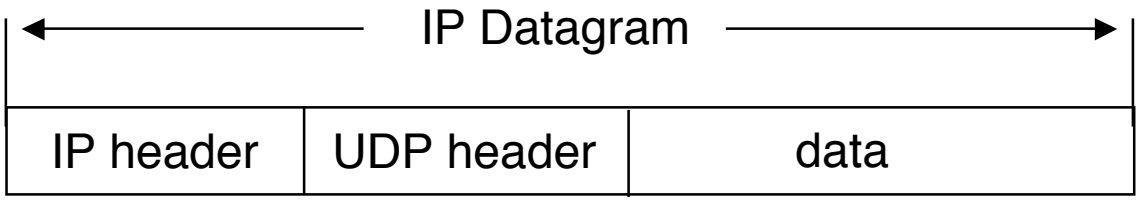
Applications	**Telnet, FTP, email, etc.**
Transport	**TCP, UDP**
Network	**IP, ICMP, IGMP**
Link	**Device drivers, interface cards**

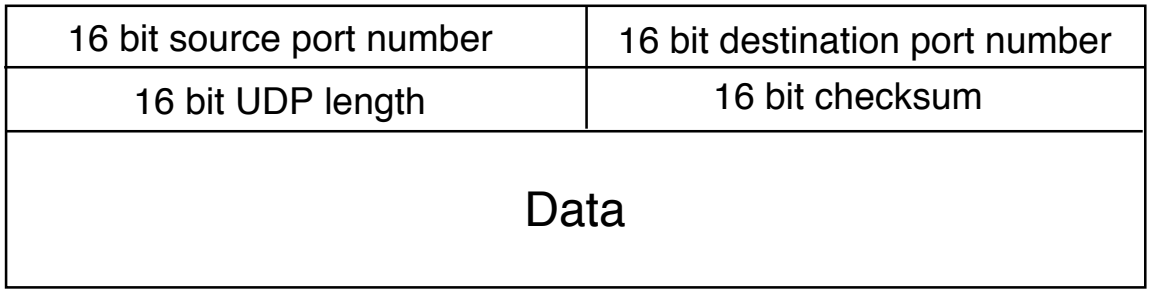
TCP - Transmission Control Protocol
UDP - User Datagram Protocol
IP - Internet Protocol

User Datagram Protocol (UDP)

- **Transport layer protocol**
  - **Delivery of messages across network**
- **Datagram oriented**
  - **Unreliable**
    - **No error control mechanism**
  - **Connectionless**
  - **Not a "stream" protocol**
- **Max packet length 65K bytes**
- **UDP checksum**
  - **Covers header and data**
  - **Optional**
    - **Can be used by applications**
- **UDP allows applications to interface directly to IP with minimal additional processing or protocol overhead**

UDP header format





- **The port numbers identifie the sending and receiving processes**
  - **I.e., FTP, email, etc..**
  - **Allow UDP to multiplex the data onto a single stream**
- **UDP length = length of packet in bytes**
  - **Minimum of 8 and maximum of 2^16 - 1 = 65,535 bytes**
- **Checksum covers header and data**
  - **Optional, UDP does not do anything with the checksum**

Transmission Control Protocol (TCP)

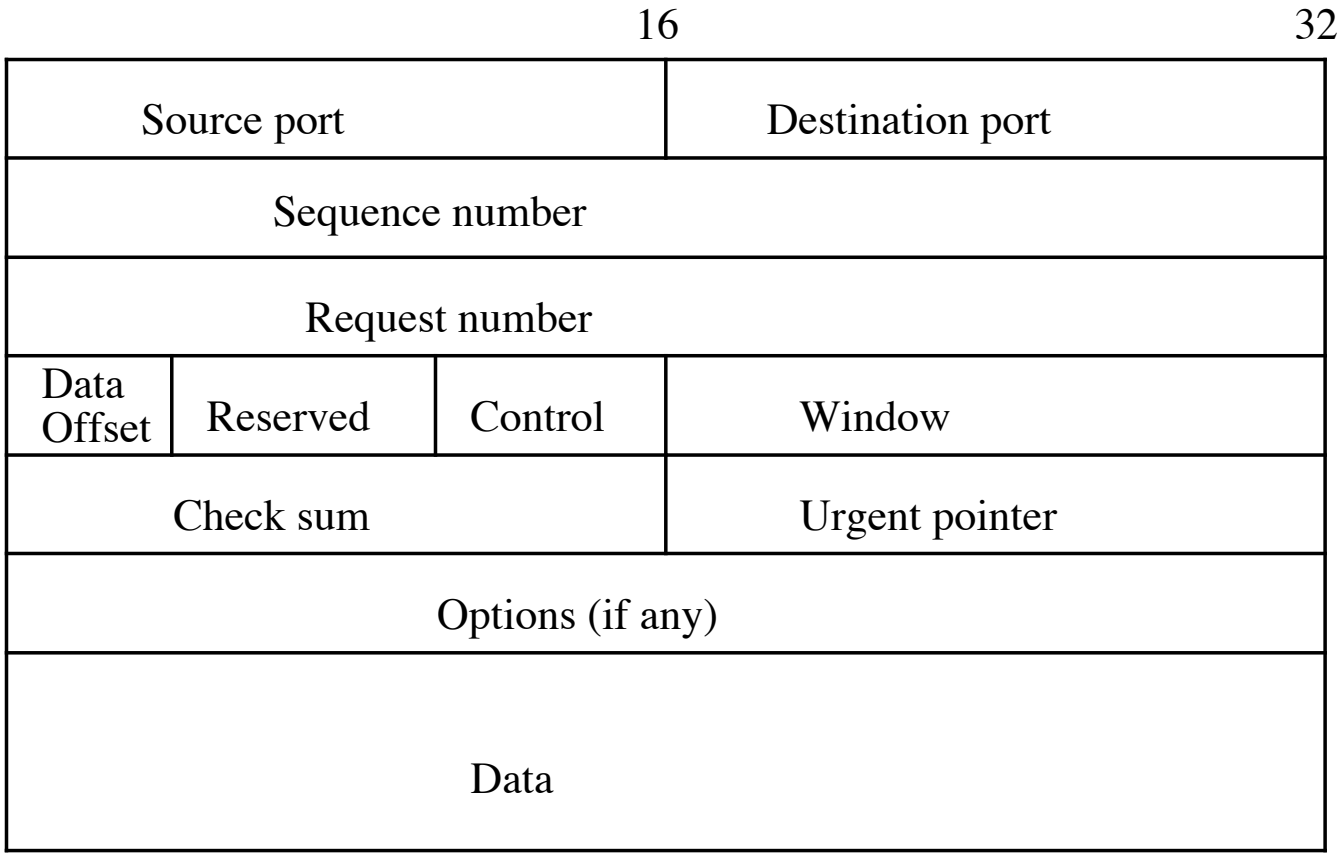
- **Transport layer protocol**
  - **Reliable transmission of messages**
- **Connection oriented**
  - **Stream traffic**
  - **Must re-sequence out of order IP packets**
- **Reliable**
  - **ARQ mechanism**
  - **Notice that packets have a sequence number and an ack number**
  - **Notice that packet header has a window size (for Go Back N)**
- **Flow control mechanism**
  - **Slow start**

    Limits the size of the window in response to congestion

Basic TCP operation

- At sender
  - Application data is broken into TCP segments
  - TCP uses a timer while waiting for an ACK of every packet
  - Un-ACK'd packets are retransmitted
- At receiver
  - Errors are detected using a checksum
  - Correctly received data is acknowledged
  - Segments are reassembled into their proper order
  - Duplicate segments are discarded
- Window based retransmission and flow control

TCP header fields

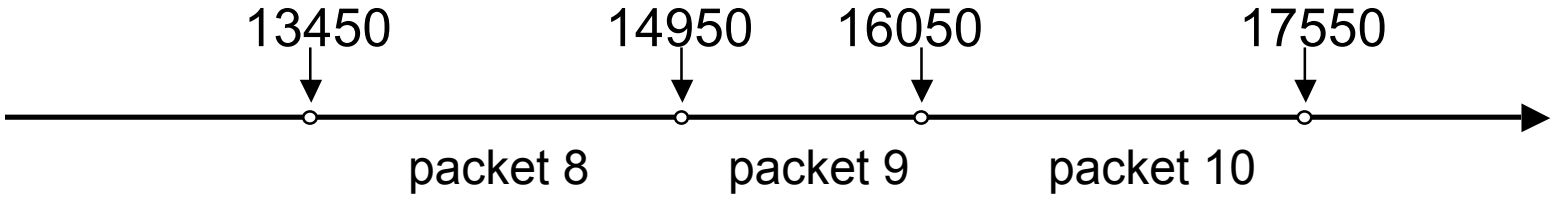


TCP header fields

- **Ports number are the same as for UDP**
- **32 bit SN uniquely identify the application data contained in the TCP segment**
  - **SN is in bytes!**
  - **It identify the first byte of data**
- **32 bit RN is used for piggybacking ACK's**
  - **RN indicates the next byte that the received is expecting**
  - **Implicit ACK for all of the bytes up to that point**
- **Data offset is a header length in 32 bit words (minimum 20 bytes)**
- **Window size**
  - **Used for error recovery (ARQ) and as a flow control mechanism**
    - **Sender cannot have more than a window of packets in the network simultaneously**
  - **Specified in bytes**
    - **Window scaling used to increase the window size in high speed networks**
- **Checksum covers the header and data**

Sequence Numbers in TCP

- **TCP regards data as a "byte-stream"**
  - **each byte in byte stream is numbered.**
    - **32 bit value, wraps around**
    - **initial values selected at start up time**
- **TCP breaks up byte stream in packets**
  - **Packet size is limited to the Maximum Segment Size (MSS)**
- **Each packet has a sequence number**
  - **seq. no of 1st byte indicates where it fits in the byte stream**
- **TCP connection is duplex**
  - **data in each direction has its own sequence numbers**



TCP Sender Events

- **Data received from application:**
  - **Create segment with sequence number**
  - **Sequence number is byte-stream number of first data byte in segment**
  - **start timer if not already running**
    - **Think of timer as for oldest un-acknowledged segment**
  - **Timer expiration interval: time-out**
- **Timeout:**
  - **retransmit segment that caused timeout**
  - **restart timer**
- **ACK received:**
  - **If acknowledges previously unACKed segments**
    - **update what is known to be ACKed**
    - **start timer if there are outstanding segments**

TCP ACK Generation

Event at Receiver	TCP Receiver Action
Arrival of in-order segment with expected seq #. All data up to expected seq # already ACKed	**Delayed ACK.** Wait up to 500ms for next segment. If no next segment, send ACK
Arrival of in-order segment with expected seq #. One other segment has ACK pending	Immediately send single **cumulative** ACK, ACKing both in-order segments
Arrival of out-of-order segment higher-than-expect seq. # . Gap detected	Immediately send **duplicate** ACK, indicating seq. # of next expected byte
Arrival of segment that partially or completely fills gap	Immediate send ACK, provided that segment starts at lower end of gap

TCP error recovery

- **Error recovery is done at multiple layers**
  - **Link, transport, application**
- **Transport layer error recovery is needed because**
  - **Packet losses can occur at network layer**
    - **E.g., buffer overflow**
  - **Some link layers may not be reliable**
- **SN and RN are used for error recovery in a similar way to Go Back N at the link layer**
  - **Large SN needed for re-sequencing out of order packets**
  - **Notice difference from Link Layer ARQ**
- **TCP uses a timeout mechanism for packet retransmission**
  - **Timeout calculation**
  - **Fast retransmission**

Retransmissions in TCP: A variation of Go-Back-N

- **Sliding window with cumulative ACKs**
  - Receiver can only return a single "ack" sequence number to the sender
  - Acknowledges all bytes with a lower sequence number
  - Starting point for retransmission
  - Duplicate ACks sent when out-of-order packet received
- **Sender only retransmits a single packet at a time**
  - Optimistic assumption: only one that it knows is lost
  - Network is congested $\Rightarrow$ shouldn't overload it
- **Error control is based on byte sequences, not packets**
  - Retransmitted packet can be different from the original lost packet (e.g., due to fragmentation)

TCP timeout calculation

- **Based on round trip time measurement (RTT)**
  - **Weighted average**

$$RTT\_AVE = a*(RTT\_measured) + (1-a)*RTT\_AVE \;\; ; ; \;\; a \sim 0.1$$

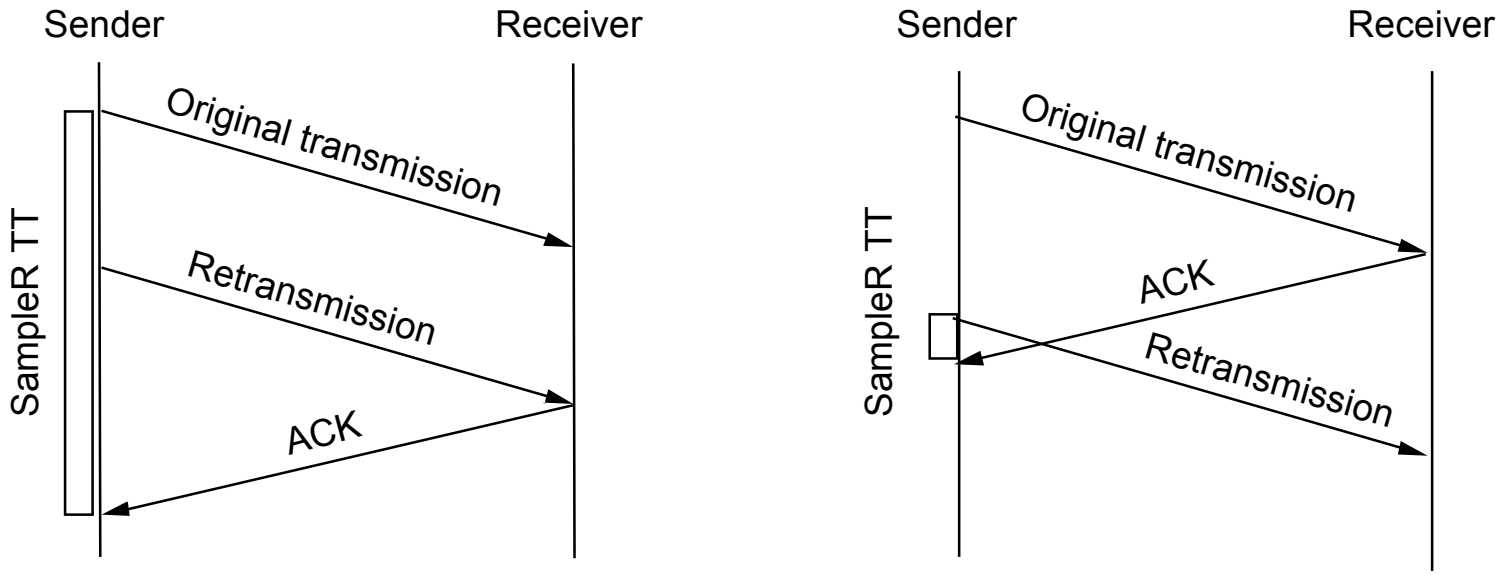
- **Timeout is a multiple of RTT_AVE (usually two)**
  - **Short Timeout would lead to too many retransmissions**
  - **Long Timeout would lead to large delays and inefficiency**

- **In order to make Timeout be more tolerant of delay variations it has been proposed (Jacobson) to set the timeout value based on the standard deviation of RTT**

$$Timeout = RTT\_AVE + 4*RTT\_SD$$

- **In many TCP implementations the minimum value of Timeout is 500 ms due to the clock granularity**

Retransmission Ambiguity & karn's Algorithm



- **Was the ACK for the original transmission or retransmission?**
  - **Wrong assumption can lead to either too large or too small a measurement**

Solution:

- **Do not sample RTT when retransmitting**
  - **only measures sample RTT for segments sent once**
- **Double timeout for each retransmission**
  - **Next timeout to be twice the last timeout, rather than basing it on the last Estimated RTT**
  - **Exponential back-off: Congestion is most likely cause of lost segments**

Fast Retransmit

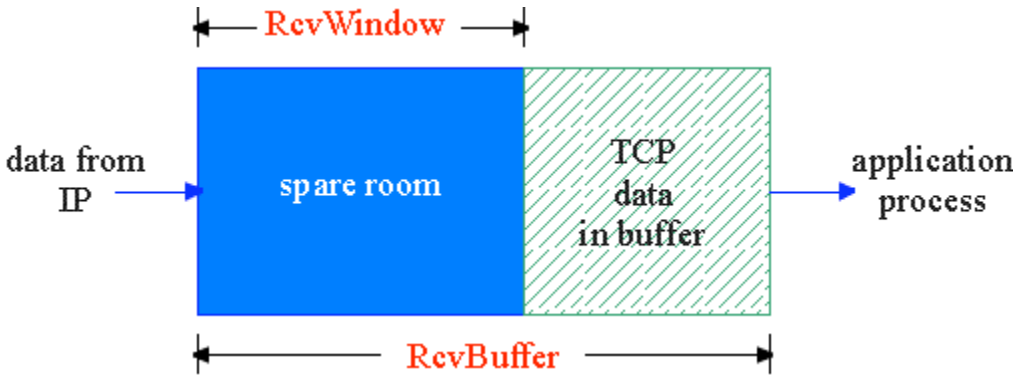
- When TCP receives a packet with a SN that is greater than the expected SN, it sends an ACK packet with a request number of the expected packet SN
  - This could be due to out-of-order delivery or packet loss
- If a packet is lost then duplicate RNs will be sent by TCP until the packet it correctly received
  - But the packet will not be retransmitted until a Timeout occurs
  - This leads to added delay and inefficiency
- Fast retransmit assumes that if 3 duplicate RNs are received by the sending module that the packet was lost
  - After 3 duplicate RNs are received the packet is retransmitted
  - After retransmission, continue to send new data
- Fast retransmit allows TCP retransmission to behave more like Selective repeat ARQ

SACK: A variation on SRP ARQ

- Option for selective ACKs (SACK) also widely deployed
- Selective acknowledgement (SACK) essentially adds a bitmask of packets received
  - Implemented as a TCP option (extended TCP header)
  - Encoded as a set of received byte ranges (max of 3 0r 4 ranges)
- When to retransmit?
  - Packets may experience different delays
  - Still need to deal with reordering
  - Wait for out of order by 3 packets

TCP Flow Control

- **Receive side of TCP connection has a receive buffer:**



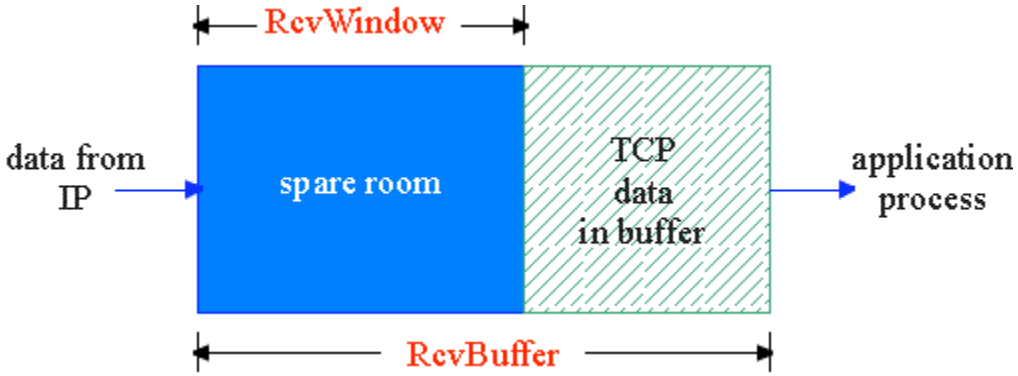
- **Application process may be slow at reading from buffer**

flow control

sender won't overflow receiver's buffer by transmitting too much, too fast

- **Speed-matching service: matching the send rate to the receiving app's drain rate**

TCP Flow Control: some details…

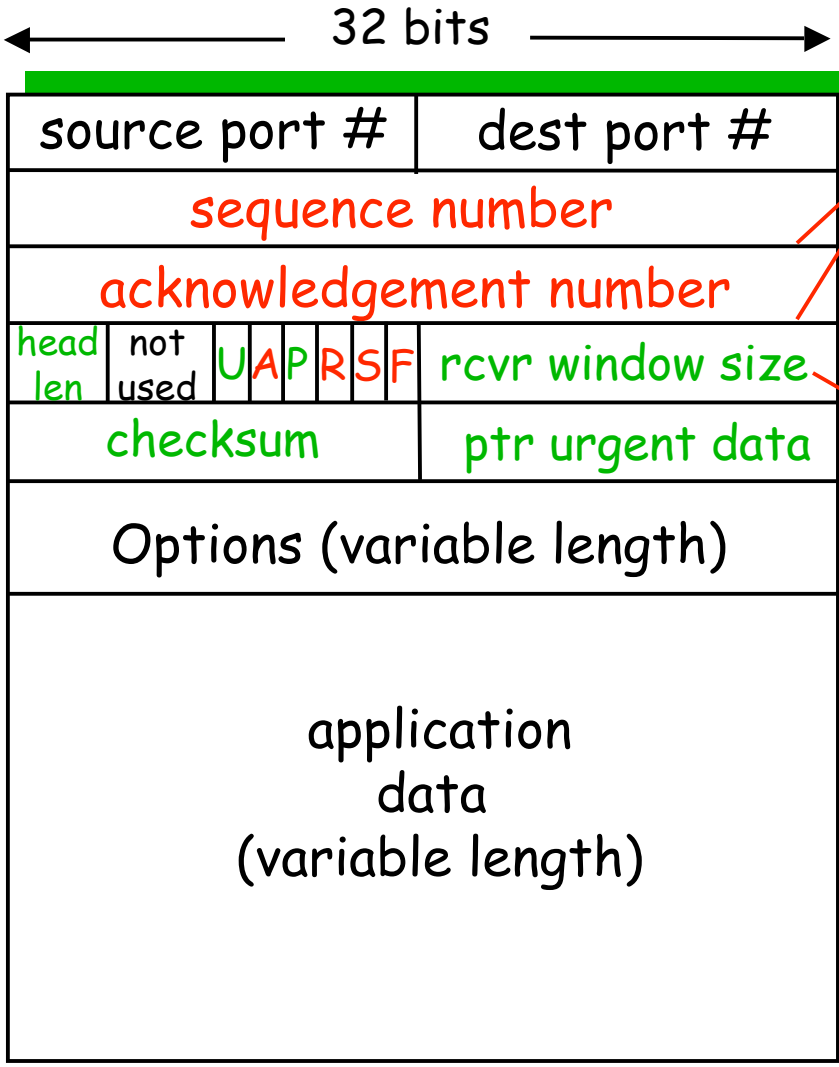


- **Suppose TCP receiver discards out-of-order segments**
- **Spare room in buffer** = `RcvWindow`

= `RcvBuffer-[LastByteRcvd - LastByteRead]`

- **Rcvr advertises spare room by including value of** `RcvWindow` **in segments**
  - **Min. Segment size**
- **Sender limits unACKed data to** `RcvWindow`
  - **guarantees receive buffer doesn't overflow**
  - **Similar to window ARQ**

TCP Segment Structure



URG: urgent data (generally not used)

ACK: ACK # valid

PSH: push data now (generally not used)

RST, SYN, FIN: connection estab (setup, teardown commands)

Internet checksum (as in UDP)

counting by bytes of data (not segments!)

bytes rcvr willing to accept

Triggering Transmission

- **How does TCP decide to transmit a segment?**
  - **Accumulated MSS (Maximum segment size) worth of data**

    Set to size of the largest segment TCP can send without local IP fragmentation (MTU of directly connected network)
  - **Sending process explicitly asked to do (Push to flush)**
  - **Firing timer (upon ack - Nagle's algorithm)**

- **Silly Window Syndrome**
  - **Very small receiver window can lead to tiny packets being sent**

Silly Window Syndrome

- **Problem:**
  - **Receiver opens window a small amount**
  - **ACK opens K < MSS bytes (very small amount of data)**
- **Should sender transmit K bytes?**
  - **Can be very inefficient as most of the packet will contain header overhead**
- **If sender is aggressive, sending available window size**
  - **Results in "silly window syndrome"**
  - **Small segment size remains indefinitely - very inefficient**

    Note that when the receiver receives the small segment,it sends back and ACK (for that small segment), opening the window for another small segment

- **Hence a problem when either sender transmits a small segment or receiver opens window a small amount**

- **Mechanism needed to wait for opportunity for sending larger amount of data**

Nagle's Algorithm

- Waiting too long hurt interactive applications (Telnet)
- Without waiting, risk of sending a bunch of tiny packets
  - silly window syndrome
- Nagle's Algorithm:
  - Continue to buffer data if some un-acknowledged packets still outstanding
  - If no outstanding data, send segment without delay
  - If more than MSS worth of data, send segment without delay
  - Additional implementation details:

    Receiver update of advertise window: avoid small increases in window size => avoid very tiny send opportunities

    Applications can disable Nagle's algorithm to avoid long delays
  - Implication: if don't have at least MSS worth of data, wait at least one RTT before transmitting new segment:

    TCP's self clocking mechanism

TCP congestion control

- **TCP uses its window size to perform end-to-end congestion control**
  - **Note difference between flow control and congestion control**
- **Basic idea**
  - **With window based ARQ the number of packets in the network cannot exceed the window size (CW)**

$$\text{Last\_byte\_sent (SN) - last\_byte\_ACK'd (RN)} \leq CW$$

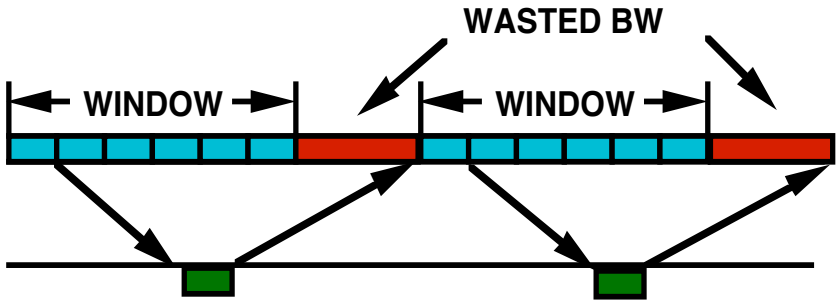
- **Transmission rate when using window flow control is equal to one window of packets every round trip time**

$$R = CW/RTT$$

- **By controlling the window size TCP effectively controls the rate**

Effect Of Window Size

- The window size is the number of bytes that are allowed to be in transport simultaneously



- Too small a window prevents continuous transmission

- To allow continuous transmission window size must exceed round-trip delay time

Keeping the pipe full
(traveling at 2/3C)

At 300 bps	1 bit = 415 miles	3000 miles = 7 bits
At 3.3 kbps	1 bit = 38 miles	3000 miles = 79 bits
At 56 kbps	1 bit = 2 miles	3000 miles = 1.5 kbits
At 1.5 Mbps	1 bit = 438 ft.	3000 miles = 36 kbits
At 150 Mbps	1 bit = 4.4 ft.	3000 miles = 3.6 Mbits
At 1 Gbps	1 bit = 8 inches	3000 miles = 240 Mbits

Dynamic adjustment of window size

- **TCP starts with CW = 1 packet and increases the window size slowly as ACK's are received**
  - **Slow start phase**
  - **Congestion avoidance phase**
- **Slow start phase**
  - **During slow start TCP increases the window by one packet for every ACK that is received**
  - **When CW = Threshold TCP goes to Congestion avoidance phase**
  - **Notice: during slow start CW doubles every round trip time**
    Exponential increase!
- **Congestion avoidance phase**
  - **During congestion avoidance TCP increases the window by one packet for every window of ACKs that it receives**
  - **Notice that during congestion avoidance CW increases by 1 every round trip time - Linear increase!**
- **TCP continues to increase CW until congestion occurs**

Reaction to congestion

- **Many variations: Tahoe, Reno, Vegas**
- **Basic idea: when congestion occurs decrease the window size**
- **There are two congestion indication mechanisms**
  - **Duplicate ACKs - could be due to temporary congestion**
  - **Timeout - more likely due to significant congestion**

- **TCP Reno - most common implementation**

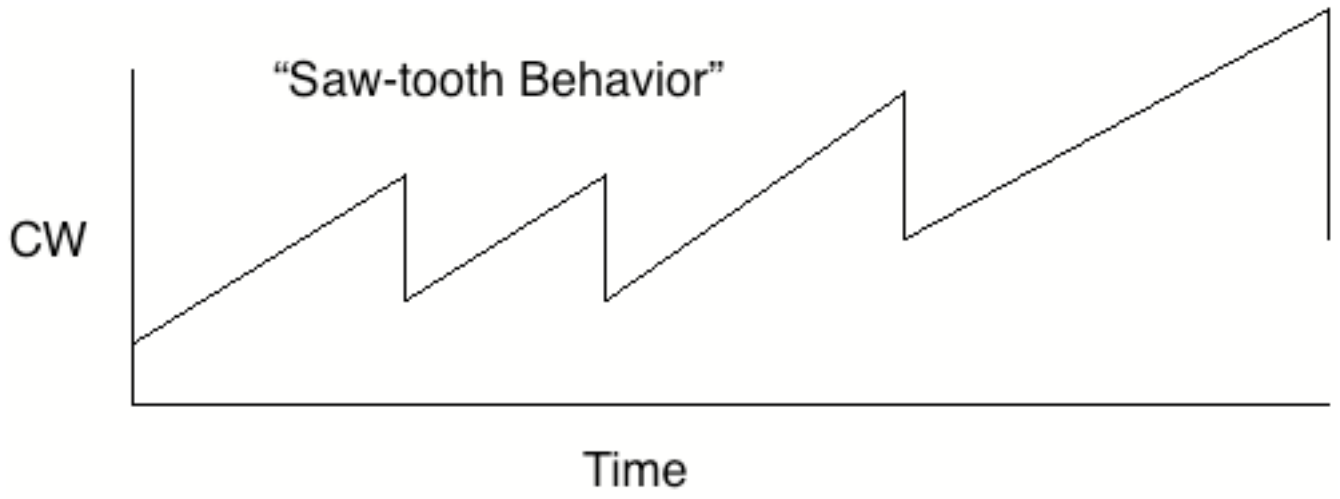
  - **If Timeout occurs, CW = 1 and go back to slow start phase**

  - **If duplicate ACKs occur CW = CW/2 stay in congestion avoidance phase**

Understanding TCP dynamics

- **Slow start phase is actually fast**
- **TCP spends most of its time in Congestion avoidance phase**
- **While in Congestion avoidance**
  - **CW increases by 1 every RTT**
  - **CW decreases by a factor of two with every loss**

"Additive Increase / Multiplicative decrease"

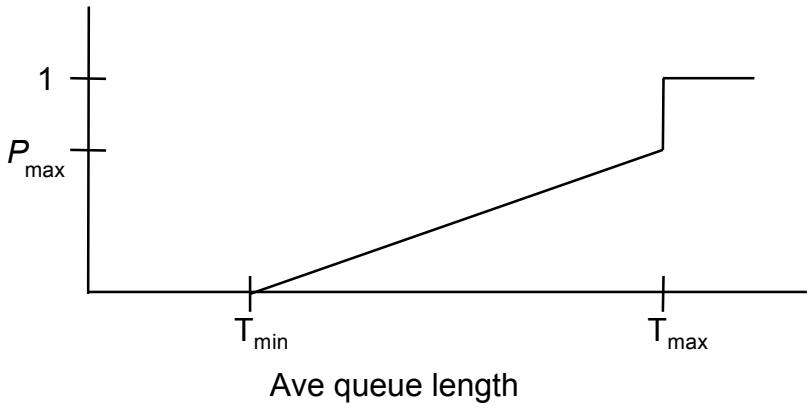


Random Early Detection (RED)

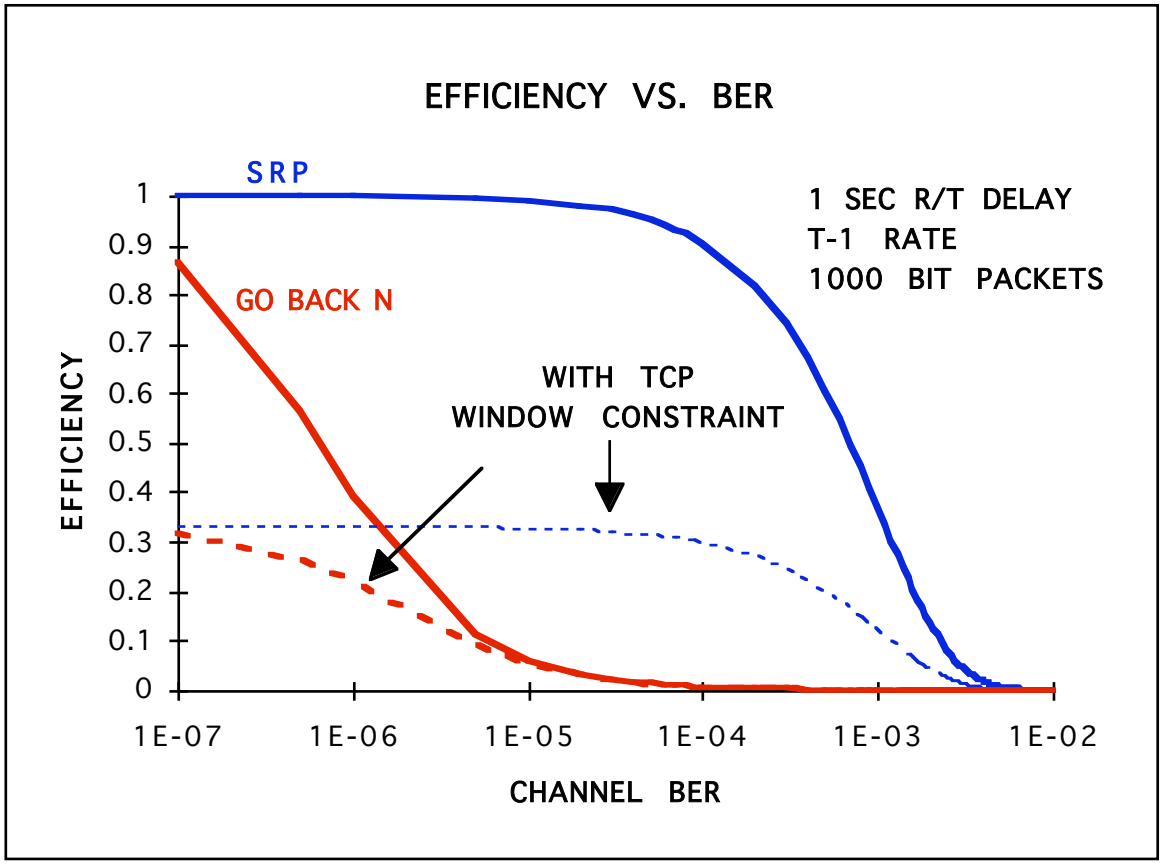
- **Instead of dropping packet on queue overflow, drop them probabilistically earlier**

- **Motivation**
  - **Dropped packets are used as a mechanism to force the source to slow down**
    - **If we wait for buffer overflow it is in fact too late and we may have to drop many packets**
    - **Leads to TCP synchronization problem where all sources slow down simultaneously**
  - **RED provides an early indication of congestion**
    - **Randomization reduces the TCP synchronization problem**

- **Mechanism**
  - **Use weighted average queue size**
    - **If AVE_Q > T_{min} drop with prob. P**
    - **If AVE_Q > T_{max} drop with prob. 1**
  - **RED can be used with explicit congestion notification rather than packet dropping**
  - **RED has a fairness property**
    - **Large flows more likely to be dropped**
  - **Threshold and drop probability values are an area of active research**

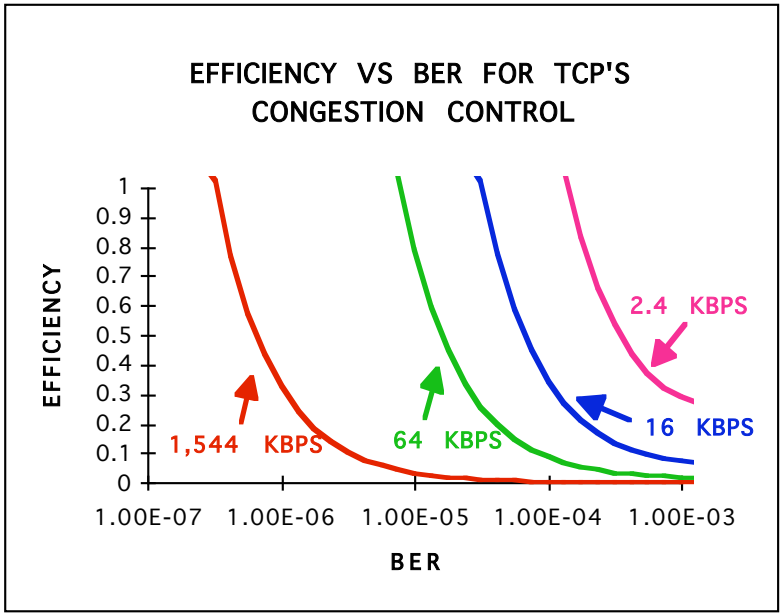


TCP Error Control



- **Original TCP designed for low BER, low delay links**
- **New implementations (RFC 1323) allow for larger windows and selective retransmissions**

Impact of transmission errors on TCP congestion control

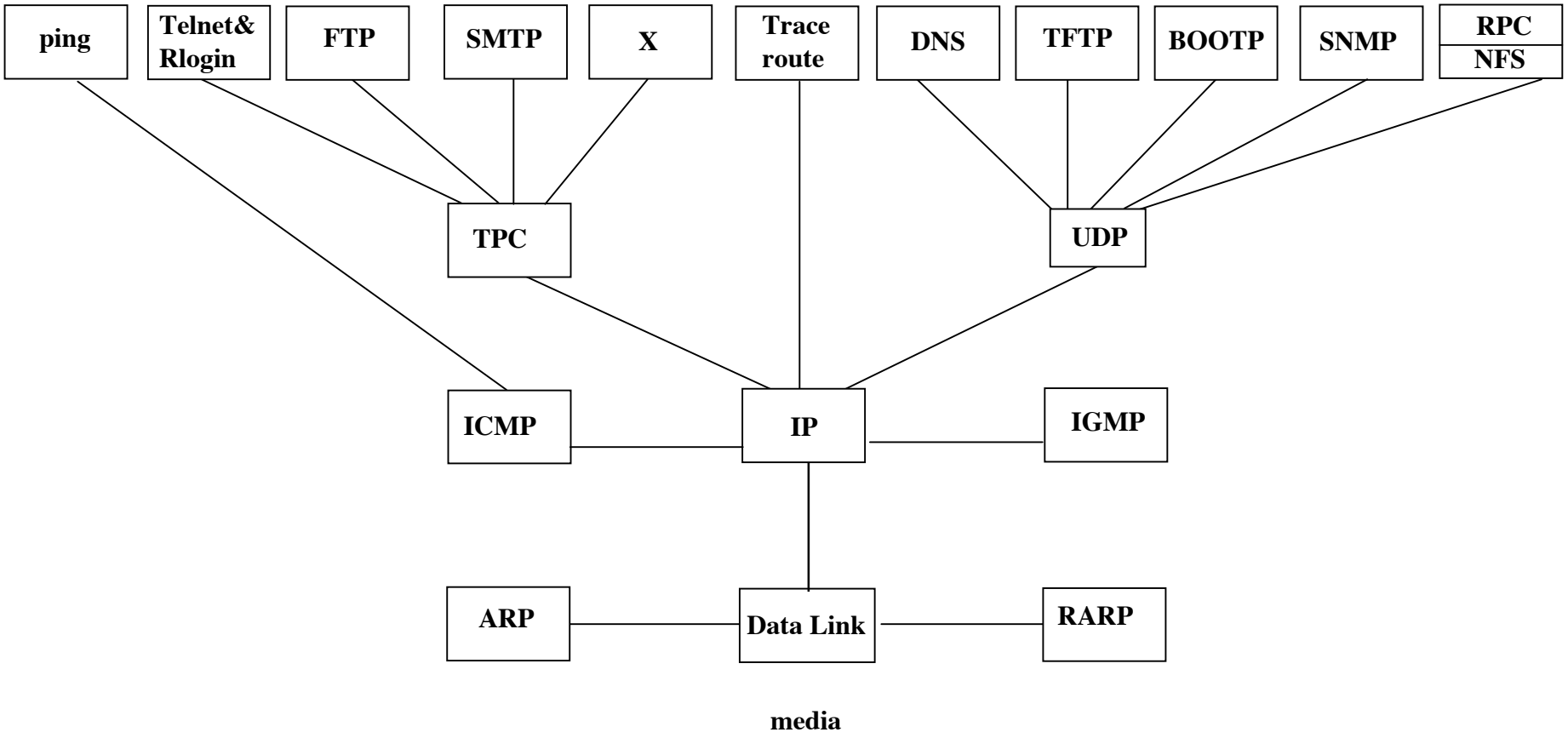


- TCP assumes dropped packets are due to congestion and responds by reducing the transmission rate
- Over a high BER link dropped packets are more likely to be due to errors than to congestion
- TCP extensions (RFC 1323)
  - Fast retransmit mechanism, fast recovery, window scaling

TCP releases

- **TCP standards are published as RFC's**
- **TCP implementations sometimes differ from one another**
  - **May not implement the latest extensions, bugs, etc.**
- **The de facto standard implementation used to be the BSD releases**
  - **Computer system Research group at UC-Berkeley**
  - **Most implementations of TCP are based on the BSD implementations**
    SUN, MS, etc.
- **BSD releases**
  - **4.2BSD - 1983**
    First widely available release
  - **4.3BSD Tahoe - 1988**
    Slow start and congestion avoidance
  - **4.3BSD Reno - 1990**
    Header compression
  - **4.4BSD - 1993**
    Multicast support, RFC 1323 for high performance
- **The BSD group is no longer in existence and new features are implemented by the various OS developers**
  - **TCP SACK, NewReno, etc.**

The TCP/IP Suite



Asynchronous Transfer Mode (ATM)

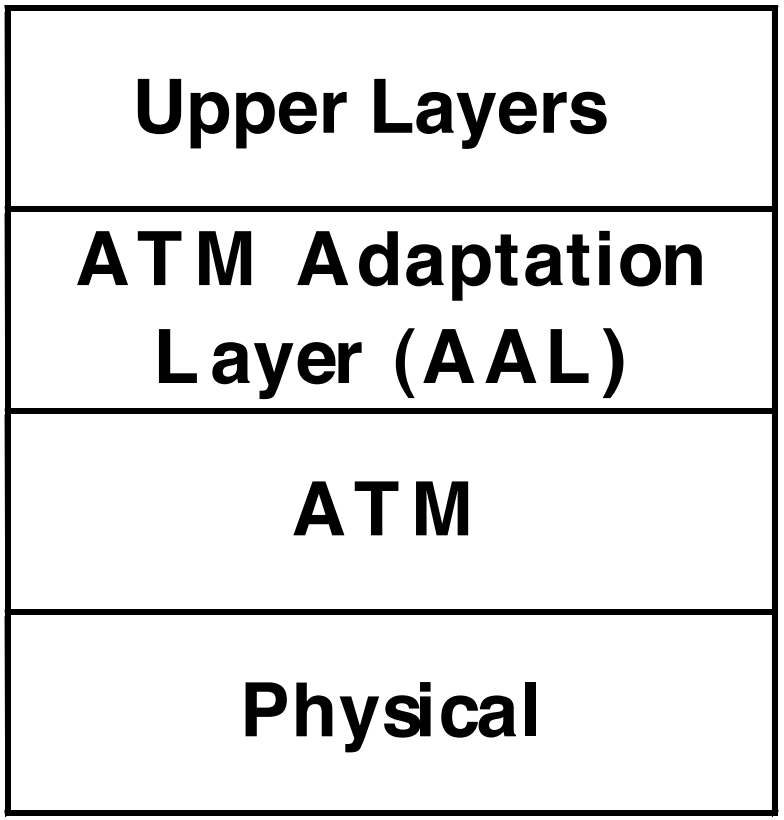
- **1980's effort by the phone companies to develop an integrated network standard (BISDN) that can support voice, data, video, etc.**

- **ATM uses small (53 Bytes) fixed size packets called "cells"**
  - **Why cells?**
    - **Cell switching has properties of both packet and circuit switching**
    - **Easier to implement high speed switches**
  - **Why 53 bytes?**
  - **Small cells are good for voice traffic (limit sampling delays)**
    - **For 64Kbps voice it takes 6 ms to fill a cell with data**

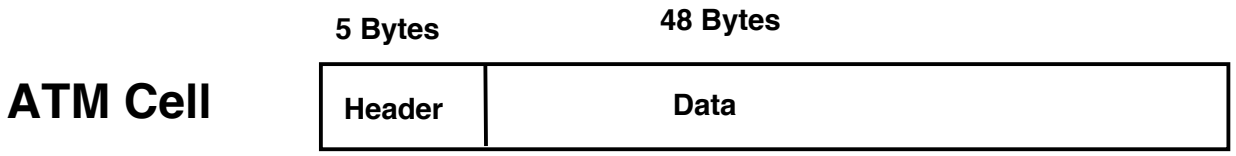
- **ATM networks are connection oriented**
  - **Virtual circuits**

ATM Reference Architecture

- **Upper layers**
  - **Applications**
  - **TCP/IP**
- **ATM adaptation layer**
  - **Similar to transport layer**
  - **Provides interface between upper layers and ATM**
    - **Break messages into cells and reassemble**
- **ATM layer**
  - **Cell switching**
  - **Congestion control**
- **Physical layer**
  - **ATM designed for SONET**
    - **Synchronous optical network**
    - **TDMA transmission scheme with 125 μs frames**

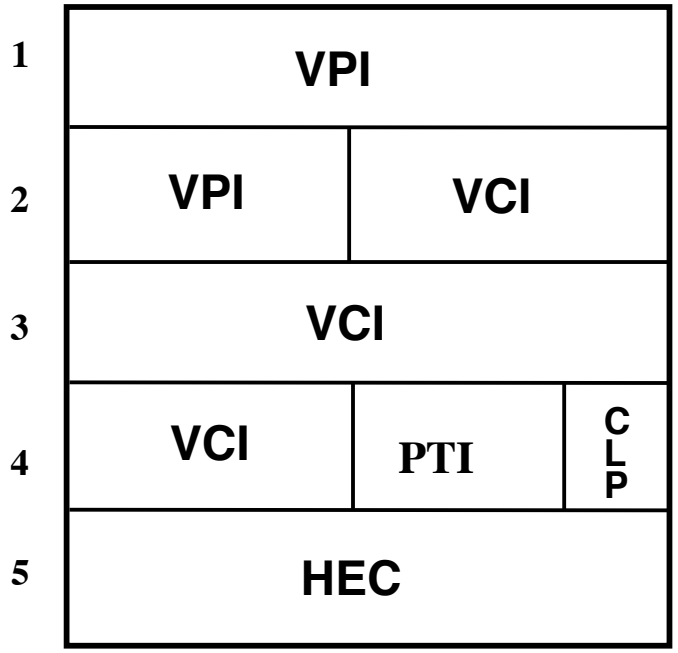


ATM Cell format



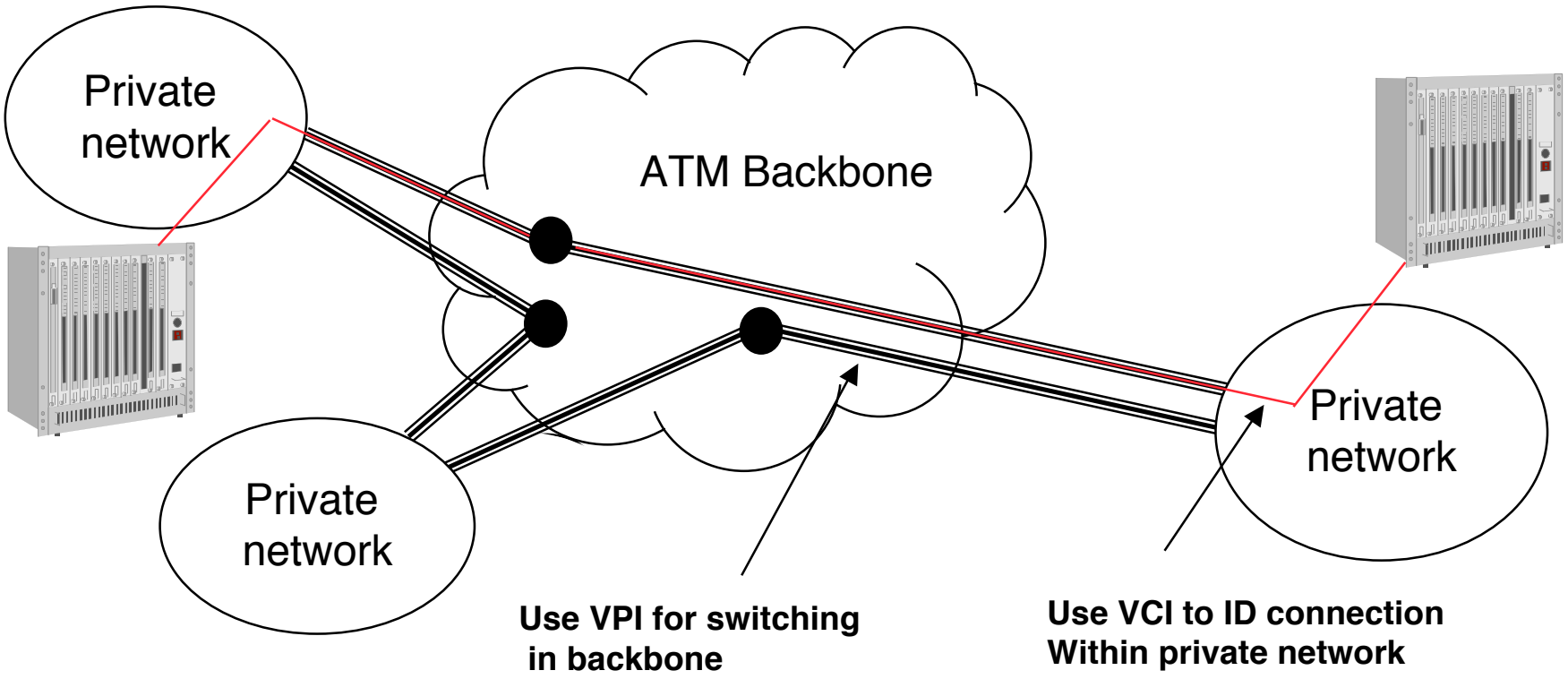
- **Virtual circuit numbers (notice relatively small address space!)**
  - **Virtual channel ID**
  - **Virtual path ID**
- **PTI - payload type**
- **CLP - cell loss priority (1 bit!)**
  - **Mark cells that can be dropped**
- **HEC - CRC on header**

ATM Header (NNI)



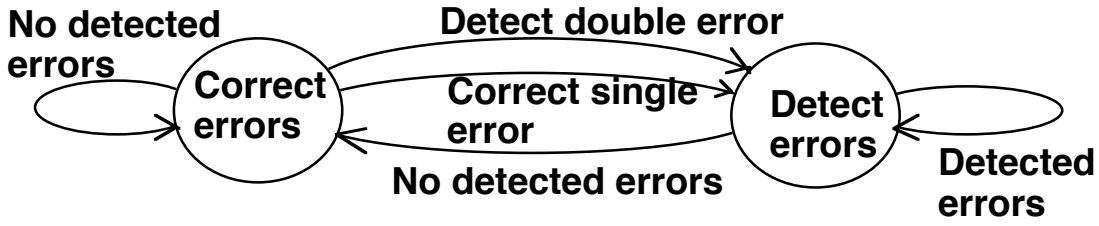
VPI/VCI

- **VPI identifies a physical path between the source and destination**
- **VCI identifies a logical connection (session) within that path**
  - **Approach allows for smaller routing tables and simplifies route computation**



ATM HEADER CRC

- ATM uses an 8 bit CRC that is able to correct 1 error
- It checks only on the header of the cell, and alternates between two modes
  - In detection mode it does not correct any errors but is able to detect more errors
  - In correction mode it can correct up to one error reliably but is less able to detect errors
- When the channel is relatively good it makes sense to be in correction mode, however when the channel is bad you want to be in detection mode to maximize the detection capability



ATM Service Categories

- **Constant Bit Rate (CBR) - e.g. uncompressed voice**
  - **Circuit emulation**
- **Variable Bit Rate (rt-VBR) - e.g. compressed video**
  - **Real-time and non-real-time**
- **Available Bit Rate (ABR) - e.g. LAN interconnect**
  - **For bursty traffic with limited BW guarantees and congestion control**
- **Unspecified Bit Rate (UBR) - e.g. Internet**
  - **ABR without BW guarantees and congestion control**

ATM service parameters (examples)

- Peak cell rate (PCR)
- Sustained cell rate (SCR)
- Maximum Burst Size (MBS)
- Minimum cell rate (MCR)
- Cell loss rate (CLR)
- Cell transmission delay (CTD)
- Cell delay variation (CDV)

- Not all parameters apply to all service categories
  - E.g., CBR specifies PCR and CDV
  - VBR specifies MBR and SCR

- Network guarantees QoS provided that the user conforms to his contract as specified by above parameters
  - When users exceed their rate network can drop those packets
  - Cell rate can be controlled using rate control scheme (leaky bucket)

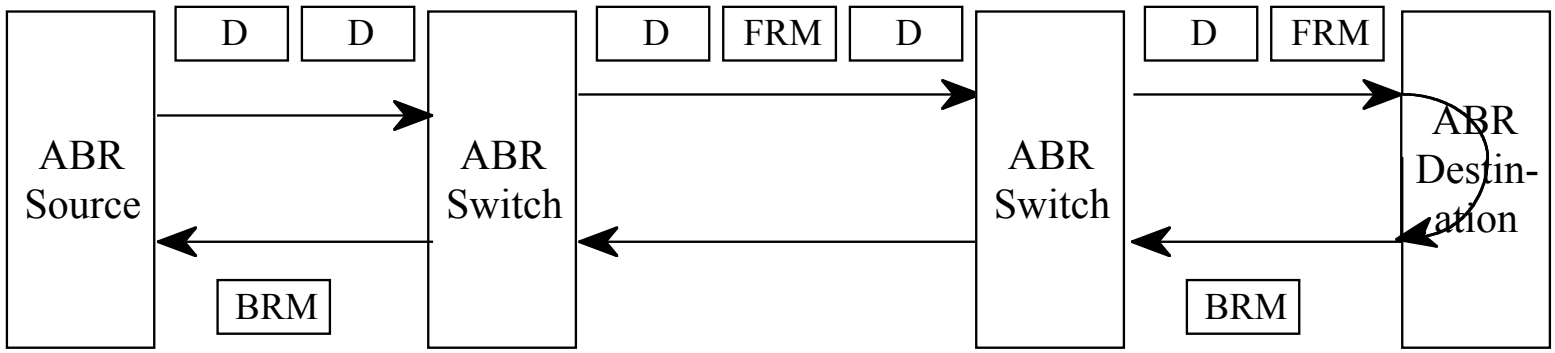
Flow control in ATM networks (ABR)

- ATM uses resource management cells to control rate parameters
  - Forward resource management (FRM)
  - Backward resource management (BRM)
- RM cells contain
  - Congestion indicator (CI)
  - No increase Indicator (NI)
  - Explicit cell rate (ER)
  - Current cell rate (CCR)
  - Min cell rate (MCR)
- Source generates RM cells regularly
  - As RM cells pass through the networked they can be marked with CI=1 to indicate congestion
  - RM cells are returned back to the source where

    CI = 1 $\Rightarrow$ decrease rate by some fraction

    CI = 0 $\Rightarrow$ Increase rate by some fraction
  - ER can be used to set explicit rate

End-to-End RM-Cell Flow



D = data cell

FRM = forward RM cell

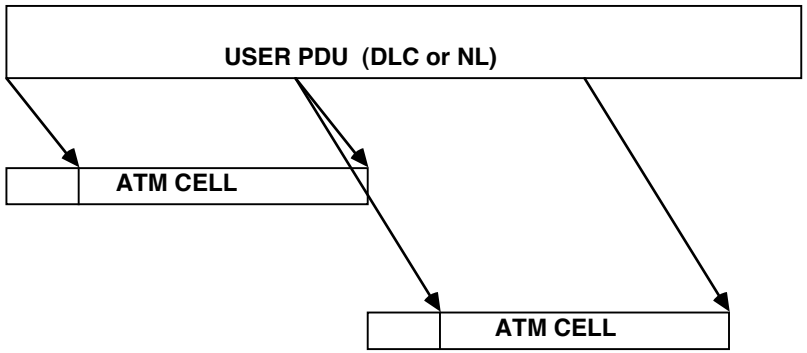
BRM = backward RM cell

At the destination the RM cell is "turned around" and sent back to the source

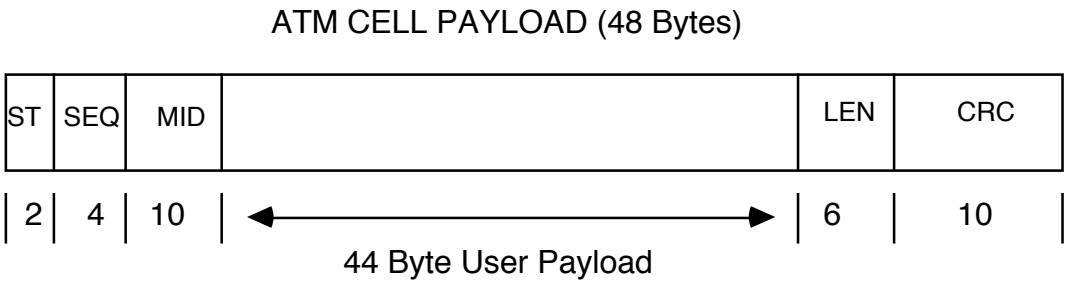
ATM Adaptation Layers

- **Interface between ATM layer and higher layer packets**
- **Four adaptation layers that closely correspond to ATM's service classes**
  - **AAL-1 to support CBR traffic**
  - **AAL-2 to support VBR traffic**
  - **AAL-3/4 to support bursty data traffic**
  - **AAL-5 to support IP with minimal overhead**
- **The functions and format of the adaptation layer depend on the class of service.**
  - **For example, stream type traffic requires sequence numbers to identify which cells have been dropped.**

Each class of service has A different header format (in addition to the 5 byte ATM header)



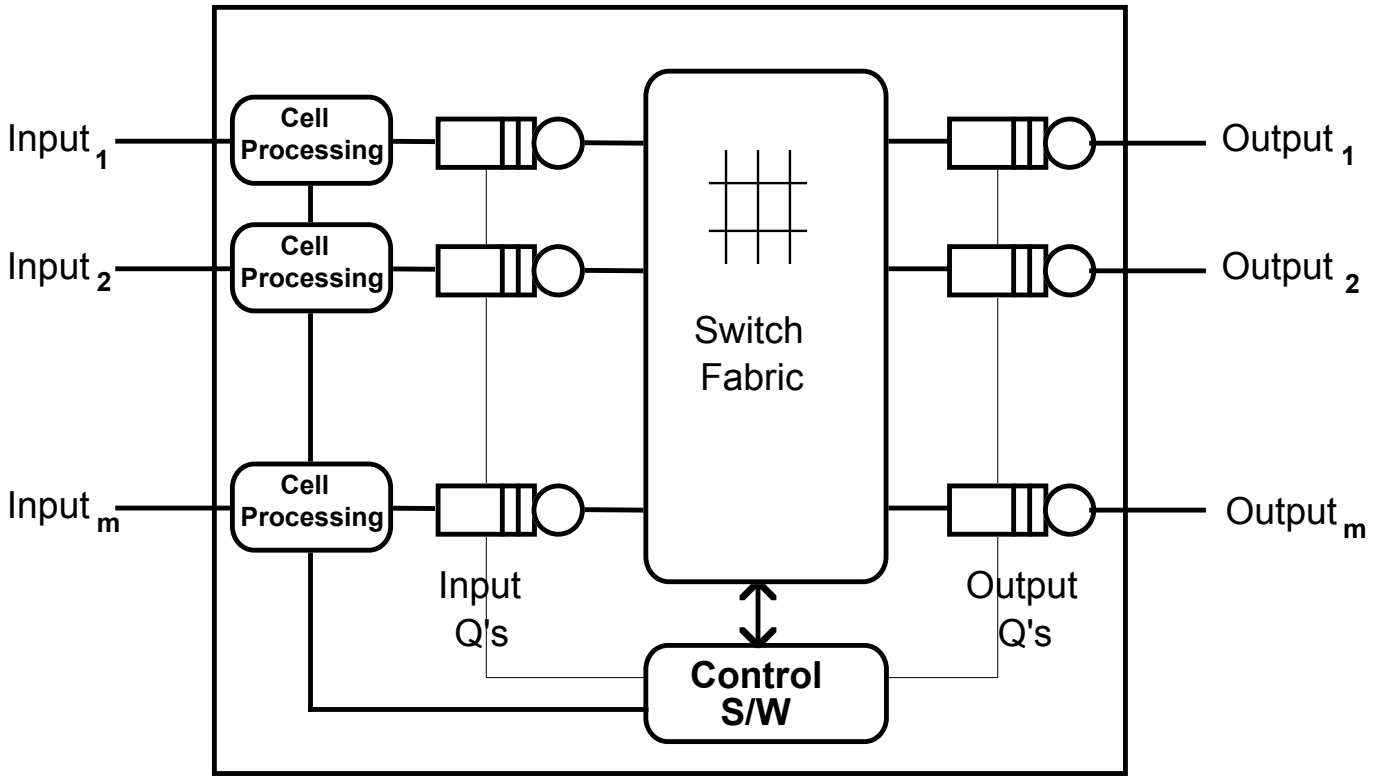
Example: AAL 3/4



- **ST: Segment Type (1st, Middle, Last)**
- **SEQ: 4-bit sequence number (detect lost cells)**
- **MID: Message ID (reassembly of multiple msgs)**
- **44 Byte user payload (~84% efficient)**
- **LEN: Length of data in this segment**
- **CRC: 10 bit segment CRC**

- **AAL 3/4 allows multiplexing, reliability, & error detection but is fairly complex to process and adds much overhead**
- **AAL 5 was introduced to support IP traffic**
  - **Fewer functions but much less overhead and complexity**

ATM cell switches



- **Design issues**
  - **Input vs. output queueing**
  - **Head of line blocking**
  - **Fabric speed**

ATM summary

- **ATM is mostly used as a "core" network technology**

- **ATM Advantages**

  - **Ability to provide QoS**
  - **Ability to do traffic management**
  - **Fast cell switching using relatively short VC numbers**

- **ATM disadvantages**
  - **It not IP - most everything was design for TCP/IP**
  - **It's not naturally an end-to-end protocol**
    - **Does not work well in heterogeneous environment**
    - **Was not design to inter-operate with other protocols**
    - **Not a good match for certain physical media (e.g., wireless)**
  - **Many of the benefits of ATM can be "borrowed" by IP**
    - **Cell switching core routers**
    - **Label switching mechanisms**

Label Switching and MPLS

- **Router makers realize that in order to increase the speed and capacity they need to adopt a mechanism similar to ATM**
  - **Switch based on a simple tag not requiring complex routing table look-ups**
  - **Use virtual circuits to manage the traffic (QoS)**
  - **Use cell switching at the core of the router**
- **First attempt: IP-switching**
  - **Routers attempt to identify flows**
    - **Define a flow based on observing a number of packets between a given source and destination (e.g., 5 packets within a second)**
  - **Map IP source-destination pairs to ATM VC's**
    - **Distributed algorithm where each router makes its own decision**
- **Multi-protocol label switching (MPLS)**
  - **Also known as Tag switching**
  - **Does not depend on ATM**
  - **Add a tag to each packet to serve as a VC number**
    - **Tags can be assigned permanently to certain paths**

Label switching can be used to create a virtual mesh with the core network

- Routers at the edge of the core network can be connected to each other using labels
- Packets arriving at an edge router can be tagged with the label to the destination edge router
  - "Tunneling"
  - Significantly simplifies routing in the core
  - Interior routers need not remember all IP prefixes of outside world
  - Allows for traffic engineering
    - Assign capacity to labels based on demand