Buffering
Outline

*Should a switch buffer?*: Forwarding modes
  - Store-and-forward
  - Cut-through

*How to buffer*
  - Memory technologies
  - FIFOs, *Where to buffer*
  - Priority queues, per-flow queueing

*What if the buffer overflows?*
  - Congestion – definition and TCP response
  - Discard policies
  - Early and Explicit Congestion Notification (separate set of slides)
Resources

Buffering: Keshav § 8.5
G. Appenzeller, I. Keslassy and N. McKeown: 'Sizing router buffers',
Proc. SIGCOMM, pp. 281-92, Aug. 2004

Partial Packet Discard: Keshav § 9.7.2

Random Early Detection:
• Keshav § 9.7.3
• RFC: 2309

Explicit Congestion Notification†:
• Keshav § 13.4.9
• RFC 3168

† Known as Explicit Forward Congestion Indication (EFCI) in ATM,
Forward ECN (FECN) in Frame Relay.
Reasons for switch buffering

To gain time: hold packets while doing things such as classification

To avoid packet loss, during times of:

- Contention: Competition for the same output. Only one packet can go out at a time; others must be buffered.
- Short-term overload (congestion): Input rate exceeds output rate; need to buffer some inputs so that they can be output later.
  - Buffering can only accommodate short-term overload.
  - In the long term, real buffers (of finite size) will overflow.

For reordering:

- e.g. to achieve TSI switching
- after mis-sequencing in the fabric (e.g. recirculating Banyan)

To reduce burstiness: To prevent overload downstream.
Goals for buffering

- Low chance of loss from overflow
- Small buffers
  - Delay
  - Implementation cost
- Simple (e.g. prefer FIFO)
Review: Statistical Multiplexing

Observations
1. The bigger the buffer, the lower the packet loss.
2. If the buffer never goes empty, the outgoing line is busy 100% of the time.
Example

- 10Gb/s linecard
  - Rule-of-thumb: 250ms of buffering (RTT)
  - Requires 300Mbytes of buffering.
  - Read and write 40 byte packet every 32ns.
- Memory technologies
  - DRAM: require 4 devices, but too slow.
- Problem gets harder at 40Gb/s
Outline

- Forwarding modes
  - Basics:
    - Store-and-forward
    - Cut-through
  - Details
Basic store-and-forward

**Store-and-forward:**
1. Switch receives whole frame
2. Stores whole frame
3. Inspects it to determine what to do next
4. Forwards it.

e.g. Ethernet frame:

```
<table>
<thead>
<tr>
<th>Destination Address</th>
<th>Source Address</th>
<th>Type</th>
<th>(Data)</th>
<th>(Pad)</th>
<th>CRC</th>
</tr>
</thead>
<tbody>
<tr>
<td>6B</td>
<td>6B</td>
<td>2B</td>
<td>0-1500B</td>
<td>0-46B</td>
<td>4B</td>
</tr>
</tbody>
</table>
```

time →

got inspect: start frame; check forwarding store it CRC & classify

Checking frame integrity

✔️ Protects against errors:
  ✔️ Header errors causing improper forwarding
  ✔️ Possibly useless frames (e.g. with error in data, leading to discard at receiver) propagating and loading network.

✖️ takes time
Basic cut-through forwarding

Cut-through:
1. Switch receives “enough” of frame to classify it (e.g. address)
2. Inspects it (e.g. address) to determine what to do next
3. If output port is free, forwards it, remaining bits will follow
   else, store it and forward when output port is free

Cut-through reduces delay, but doesn’t protect against errors
Types of erroneous frames

What types of frames can store-and-forward protect against?

Frames with bit errors (lack integrity) usually detected by CRC mismatch.

Frames with incorrect length:

- **Runts**: too small
  - Most protocols have frame overheads (e.g. 802.3†: 18B, 802.11: 28B)
    ⇒ frames shorter than minimum overhead are invalid
  - Ethernet imposes minimum frame length (64B, excluding preamble) to ensure that source can hear any collision while it is transmitting the frame

- **Giants**: too large
  - Protocols limit frame length to prevent large serialisation delays during periods of contention.
    e.g. Ethernet payload ≤ 1500B ⇒ frame > 1518B is invalid

† Only MAC overheads are shown. Physical layer preambles are also overheads.

Store-and-forward details

• Can start classification as soon as required info received, e.g. progress in parallel with receipt of payload (no need for delay after frame), and only leave decision of whether to forward to end of frame.

✓ Confines propagation of erroneous frames, in particular giant frames or those without integrity.
⇒ won’t waste transmission capacity forwarding frames that will ultimately be discarded by the receiver anyhow.
saves transmission capacity by sacrificing delay

aka “receive-and-forward”
Cut-through forwarding details

✓ Can reduce the forwarding delay across the switch

\[
delay_{\text{min}}(\text{store-and-forward}) = \frac{\text{frame length}}{\text{transmission rate}}
\]
e.g. 1.2ms for 10Mb/s Ethernet and 1.5kB frame

\[
delay_{\text{min}}(\text{cut-through}) = \frac{\text{DA length}}{\text{transmission rate}} + \text{classification delay}
\]
e.g. 4.8us for 10Mb/s Ethernet + classification delay

May become important when delays accumulate as path traverses multiple switches

Alternative technique is to reduce frame size, e.g. ATM

Note that “cut-through forwarding” is distinct from “cut-through routing” [Keshav, p. 537]
Also, “cut-through forwarding” is distinct from “buffer cut-through” in which layers pass pointers to data between themselves, rather than the data itself.
Limitations of cut-through forwarding

- May have to store frame anyhow if output port is busy
  - Subject to MAC, e.g. CSMA/CD access delays
  - Possible only when there is no queuing at the output port
- Difficult when input rate < output rate†
  e.g. frame of 1kB takes 800us @ 10Mb/s, and 80us @ 100Mb/s
  if start forwarding frame at time 4.8us will run out of data to send
  input volume $48b + 10M \times t = output volume 100M \times t$.
  equal after 0.53us (at 53“.3.” bits)
  ⇒ need to wait (720us) to receive enough data from input to sustain output (dashed line)
- Difficult when transmitting on multiple output ports (e.g. multicast)
  - some ports may be busy while others aren’t
  - output ports may have different rates
  ⇒ “cut-through is not an alternative to store-and-forward operation, it is in addition to it.”

† Cisco terms: “synchronous switching”: forwarding packets between ports operating at same speed, and “asynchronous switching” if ports have differing speeds.
Variations of cut-through

“fast forward”: Cisco term for plain cut-through (to distinguish it from ...)

“fragment-free cut-through”:
- ensures frame exceeds minimum length (little additional delay)
- Cuts-through once this condition is met, preventing propagation of runt frames.

**Adaptive:** Many switches operate in an adaptive mode:
1. Measure frame error rates
   (i.e. check CRCs even with cut-through – too late to abort, but may affect mode)
2. If error rate is low (e.g. <10%†)
   use cut-through: low delay, few erroneous frames propagating
   else
   use store-and-forward: increases delay but confines erroneous frames

† Default value for Cisco Catalyst 2900 switch
Cut-through switches verify frame integrity anyhow s.t. bridge learning process doesn’t incorrectly learn source locations.

Copyright © 2008 Tim Moors
Variations of cut-through

Figure from CCNA course material.

Adaptive cut-through checks the errors on each port and senses the best forwarding mode.
Cut-through & layer of operation

“Switches can cut-through, but routers can’t. So switches are faster”

**Theory**: Layers are separate, and higher layer doesn’t start processing incoming packet until lower layer has finished with it.
⇒ Interpretation of network layer (IP) fields can’t start until frame has been fully received and link layer has checked integrity.

**Practice**:
- Switches at low layers may interpret fields from higher layers. e.g. frame switch may inspect IP address to determine outgoing port.
- Violates insulation benefits of layering, but provides marketing advantage (network layer routing, without high delays).
Summary of forwarding modes

Store-and-forward:
• Receives whole frame before starting to forward it.
• Checks integrity before forwarding.
  • Can prevent erroneous frames being forwarded.

Cut-through:
• Can start forwarding as soon as packet classified and output is free
  • Potentially low delay.
  • May have to store-and-forward if output isn’t free.
• Checks integrity while forwarding.
  • Can’t prevent erroneous frames being forwarded
  • But can monitor error rate. Adaptive system will switch to store-and-forward if past error rate is high.
Outline
Basic memory device options

Want:
• Multiple ports (switch input and output); preferably concurrent access
• Fast I/O of frames (sequences of bits/words)
• Random access to memory content (e.g. for switch to inspect address fields)

Multi-port memories
• Buffer (e.g. FIFO) devices, e.g. Dual-ported RAM
• Video RAM →

! Bonus mark to anyone who can provide
• circuit diagrams for each type of memory (real DPRAM is elusive)
DRAMs vs SRAMs ⇒ VRAMs

Conventional single-port memories
- Static RAM (SRAM)
- Dynamic RAM (DRAM)

Performance comparison:
- Access speed: Fast SRAM vs slow DRAM
- Power consumption: High for SRAM vs low for DRAM
- Capacity: High for DRAM (small 1 transistor/bit cells), low for SRAM (6 transistors/bit)
  - True multiport memories have even lower density – e.g. 8 transistors/bit

Constraint: Packaging limits pin count
  => narrow intra-chip buses vs potentially massive intra-chip buses

The state of the art for commercially-available chips (not modules) in 2007: 32Mb @ 250MHz for static RAMs (e.g. uPD44323362) and 512Mb @ 200MHz for SDRAMs (where the S is for Synchronous, not Static). Data can be accessed on both rising and falling edges of the clock, leading to data rates twice the clock rate.
Video RAM

Video application needs:

- Multiport memory:
  - 1 random-access port for CPU access
  - 1 serial port read to feed Cathode Ray Tube (CRT)
- Large size: \(1024 \times 768\) pixels \(\times 16b = 12.6\text{Mb}\) dual port
- High speed: \(12.6\text{Mb} @ 24\text{ frames/sec} = 0.3\text{Gb/s}\)

(Less relevant with modern LCD displays)
Implementing Video RAM

- DRAM provides large size
- On-chip copy to separate memory ("cache") allows concurrent CPU&CRT access
  - On-chip: exploits wide on-chip buses, e.g. 2048b
  - Separate memory:
    - Sequential access suffices (avoiding random access saves pins)
    - Often implemented as SRAM for speed

Serial clock lines are inputs to => really should point into the chip
VRAM for networking

Networking has similar requirements to video: High-speed sequential access to large multiport memory

Implementation:
- Ports access caches. Easy to expand caches from 2-port video RAM to p-port network RAM
- Achieve shared-memory switching by using DRAM to transfer between caches & for buffering
- Bit masks indicate which bits of row to write; e.g. if storing multiple packets per row.
Example of a VRAM switch: IDT 77V400 chip
First In First Out queues (FIFOs)

- The simplest type of buffer because output order = input order
- Readily handles variable-length packets
  - May be implemented:
    - using a RAM as a circular buffer
    - as a dual-port memory chip (e.g. IDT7028L: 64K x 16)
    - optical delay line
- Discard policy is limited to:
  - Drop-Tail (don’t add to FIFO) or
  - Drop-Head (discard first out)

FIFOs are very similar to First Come First Served (FCFS) queueing:
When a FCFS queue is served by a single server & all jobs have the same service time, the result will be FIFO
Locating FIFOs

- **Input ports**: Resolve contention by queueing at input and sending through fabric when output is free.

- **Output ports**: FIFOs are fine. Reliance on output requires fast switch and buffers since many inputs may simultaneously flow to one output. May need some output buffering because of variable delay in accessing shared medium.

- **Within the switch fabric**

Non-FIFO buffers

Non-FIFO is difficult:

- Need to maintain a **list showing packet service order**
- With variable-length packets: Tradeoff between wasting space and having space scattered in useless positions.
  - Solution 1: Packets occupy contiguous bytes of memory according to length. If multiple short packets leave buffer, may have enough bytes to store a new long packet, but not in contiguous positions.
    ⇒ expensive shuffling for “garbage collection”
  - Solution 2: Packets occupy fixed-length pages of memory.
    - One page size: Simple, but too large for some packets
      ⇒ waste space
    - Multiple page sizes: Complex & might only have many small ones free. Only need a few different sizes (e.g. short for TCP ACK or long for TCP MSS)
  - Solution 2’: Packet is fragmented into pieces, each stored in a fixed-length page of memory, **linked in a list**.
- Easier for fixed-length packets (ATM – may waste transmission capacity, or fixed length within switch)
Priority queues

Separate queues (e.g. linked lists) for different packets having different delay requirements, e.g. voice: high priority, file transfer low priority.

Service disciplines:
• Preemptive: *Empty high priority queue before* serving low priority queue. New high priority arrivals *preempt* existing low priority jobs.
• Weighted: *Serve high priority queue more often* than low priority queue. High priority gets *better* service, but not exclusive service.
Per-flow\textsuperscript{†} queueing

We seek some *isolation* between flows of information (from one source to another) s.t. if one transmits excessively fast, it won’t exhaust the buffer and lock out others.

FIFO provides no isolation: Service is proportional to arrival rate.

⇒ Ideally switch should have a separate queue for each flow

**Identifying flows** is:

- simple when the switch maintains state about connections (e.g. VCs):
  - Classify packet and connection state will record whether connection has recently sent a burst.
  - Store packet only if connection has been well-behaved.
  ⇒ another advantage of virtual circuits.
- hard when the switch lacks state (e.g. IP routers): No record of past activity

⇒ For IP routers, prefer congestion control techniques that don’t need to identify flows and can use FIFO queuing

\textsuperscript{†} aka per-connection queueing
Outline
**Congestion**

Definition: When the network service is poor because of excessively high load.

Informally: “too many sources sending too much data too fast for the network to handle”

Symptoms:
- Packet loss (load>capacity)
- Delays (load near capacity)

**Congestion collapse:** Network throughput deteriorates with increasing load:
- Packet loss causes retransmission
- Packet delays cause unnecessary retransmission.

Both magnify the load on links near sources

![Figure from Kurose and Ross](image)
Transmission Control Protocol

- TCP provides reliable unicast transfer for applications like web and email access
- The early Internet lacked mechanisms to deal with congestion, leading to episodes of congestion collapse in the late 1980s.
- Too hard to change routers (often implemented in hardware)
  \( \Rightarrow \) change TCP (implemented in software & BSD Unix was widely used)
**TCP’s congestion control**

- Use packet loss as a congestion indicator
- Increase rate linearly over time when there are no indications of congestion (to approach, and eventually surpass available capacity)
  - TCP also includes a Slow Start phase to expedite the increase when well away from congestion level, consisting of limited bursts of packets.
- Reduce rate multiplicatively over time (e.g. halve it) when source observes indications of congestion.

i.e. TCP responds to packet loss by slowing down.

Overload isn’t detected immediately because of router buffers, round trip times, and delayed timeout.
Review: TCP Congestion Control

Only $W$ packets may be outstanding

Rule for adjusting $W$:
- If an ACK is received: $W \leftarrow W + \frac{1}{W}$
- If a packet is lost: $W \leftarrow \frac{W}{2}$

Slide from Nick McKeown, based on animations from http://yuba.stanford.edu/~appenz/animations.html
Outline

Discard strategies

• According to position in queue
• Loss priorities
  • Source marking
  • Network marking
• Discarding when buffers aren’t full:
  Partial Packet Discard
  Early Packet Discard
  Random Early Detection

Explicit Congestion Notification – an alternative to discard
Dropping old or new packets

Some things get better with age, some get worse.

Analogy often drawn with milk vs wine.

- Milk: Packets that get *worse* with age: Multimedia: Packet may arrive too late to be played out in time.
- Wine: Packets that get *better* with age (within limitations): Packets involved in reliable transfer: If using go-back-$N$ retransmission, the smaller $N$ is, the more that will need to be retransmitted.

i.e. switch should drop:

- **drop tail** for TCP: Drop newest packet (TCP provides reliable transfer; though not necessarily go-back-$N$)
- **drop head** for UDP: Drop oldest (UDP often carries multimedia)

i.e. switch might inspect IP protocol number to determine whether TCP or UDP
Loss priorities: Source tagging

Some applications exchange packets that vary in their loss sensitivity.

e.g. streaming media can often be coded hierarchically:
   one level of low-fidelity baseband information
   + one or more levels of enhancement information (less loss sensitive)
e.g. standard definition TV + high definition TV supplement.

Such applications benefit from the network discarding unimportant packets (e.g. HDTV) rather than important packets (e.g. SDTV).

Sources have incentive to “tag” packets, indicating their importance (loss sensitivity). “Importance” is only relative to other packets from this application.
Implementing tagging

**ATM**: Cell Loss Priority bit in cell header

**IPv4**: TOS (prior to diffserv reuse of this field)

\[
\text{R} = \text{reliability (1=lower loss, 0=higher loss)}
\]

---

**IPv6**: set the Traffic Class/Differentiated Services field to indicate the required per hop behaviour (drop priority)
Network tagging of loss priority

To guarantee service: During Call Admission Control, applications negotiate a contract with the network.
- Network agrees to provide service guarantees.
- Application agrees to abide by certain traffic profile (e.g. as described by a Leaky Bucket).

Network may “police” source traffic to ensure that it conforms to the agreed profile.
- Traffic that does not conform to the profile may be immediately discarded, or may be tagged for preferential discard within the network.
- Source can “get away” with transmitting excess traffic, but only if it doesn’t degrade service to others.

... end of tagging discussion.
Review: Segmentation and Reassembly

Many layers *segment* the Service Data Unit that they receive from the higher layer into multiple smaller Protocol Data Units.

* e.g. 1: Network layer may fragment a large amount of data (e.g. 9KB) supplied by the transport layer† so that fragments can be transmitted over a link with a smaller Maximum Transmission Unit (e.g. Ethernet with 1500B maximum frame length needs 9KB to be fragmented into 6 parts).

* e.g. 2: Asynchronous Transfer Mode: Uses short (48B of data) cells in order to limit voice packetisation delays. If Ethernet frame is to be sent over an ATM link connecting routers then it needs to be segmented (e.g. 1500B Ethernet frame → 32 cells).

We’ll refer to the large SDU as the “whole” and the smaller PDUs as “parts”.

For generality, without confusion about segments (=parts in ATM, = whole for TCP/IP), fragments, packets, frames, cells, ...

† TCP may apply a Path MTU discovery process, which may render network layer fragmentation unnecessary.
Partial Packet Discard

The smaller the data unit, the higher the percentage overhead for encapsulation fields such as sequence numbers and checksums.

⇒ “parts” often just include addressing, and omit reliable transfer fields.

⇒ Reliable transfer only operates on the “whole”.

If any “part” is lost, must retransmit the “whole”

  e.g. TCP retransmits “whole” (segment) when one IP “part” (fragment) is lost

⇒ If router discards one part of the whole, then it might as well discard all ensuing parts: they will be retransmitted anyhow.

✓ Good policy for applications requiring complete transfer: Releases space in router’s buffer

✗ Bad policy for applications that tolerate small loss – magnifies the loss
Outline

- ECN continued in another slide set