C:\cygwin\home\zahorjan\cse461\12au\03-framing.cp3

CSE 461 – Framing

Recap and Plan

We've seen that:

- Signals are used to carry bits across links as symbols
- We can code enough transitions to find the symbol boundaries

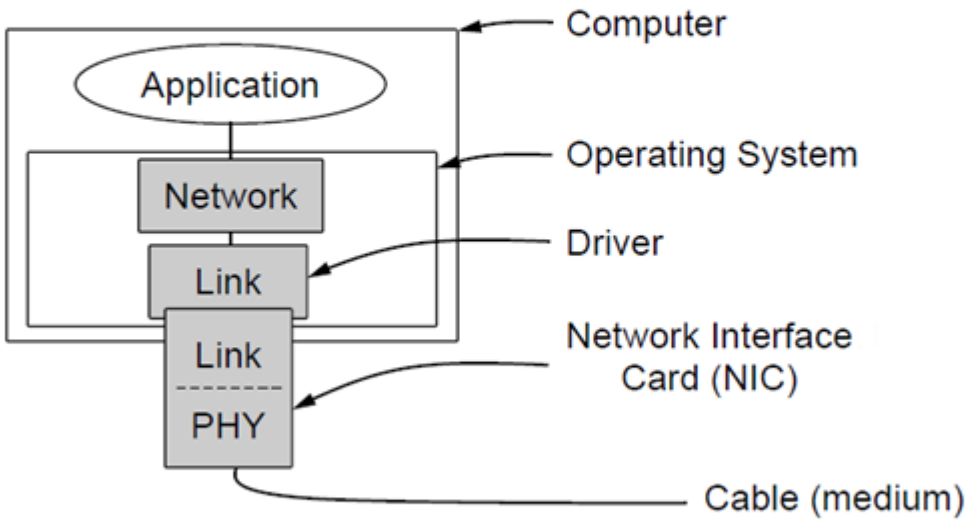
We want to send digital messages across links

1. Need to find the start/end of messages (framing)
2. Need to find and fix any transmission errors (codes, ARQ)

End result is an abstract link

- Sends messages up to M bytes, at R bit/sec, with delay D sec

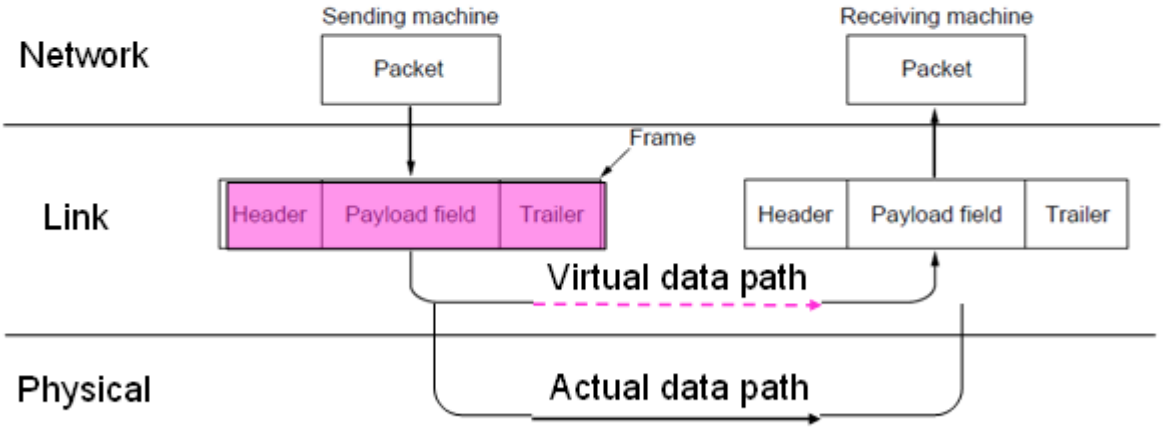
Typical implementation of the layers



3

Frames

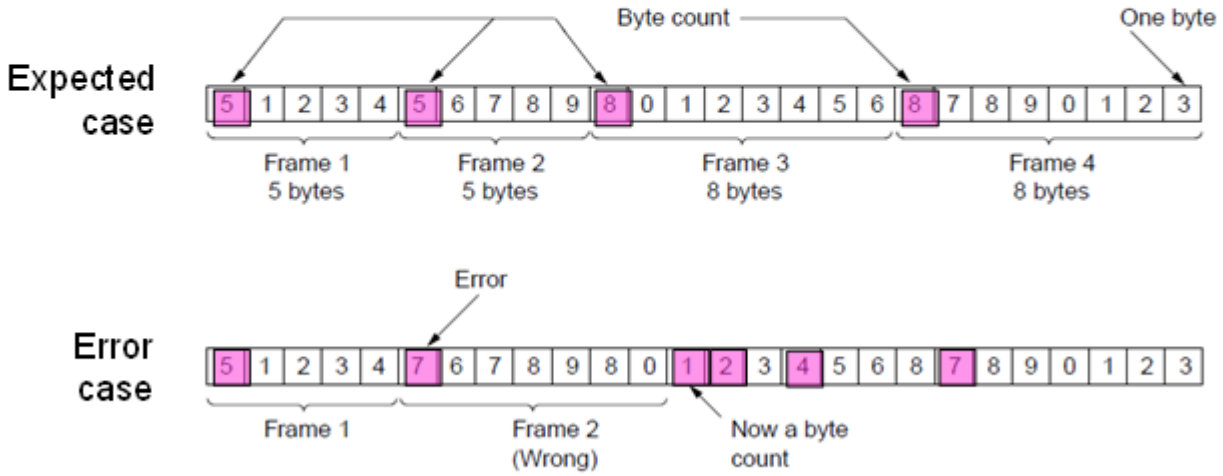
Link layer accepts *packets* from the network layer, and encapsulates them into *frames* that it sends using the physical layer; reception is the opposite process



Framing: Byte count

Frame begins with a count of the number of bytes in it

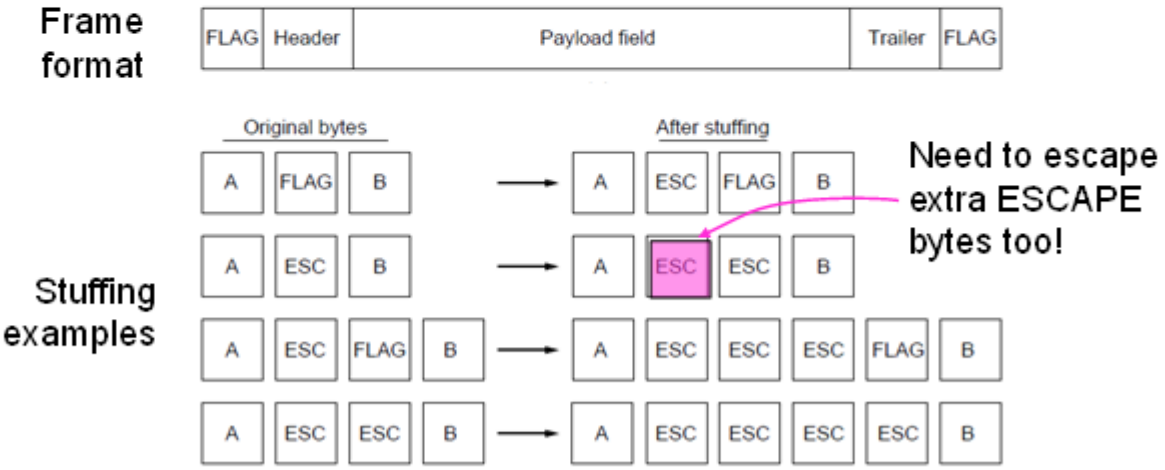
- Simple, but difficult to resynchronize after an error



Framing: Byte stuffing

Special flag bytes delimit frames; occurrences of flags in the data must be stuffed (escaped)

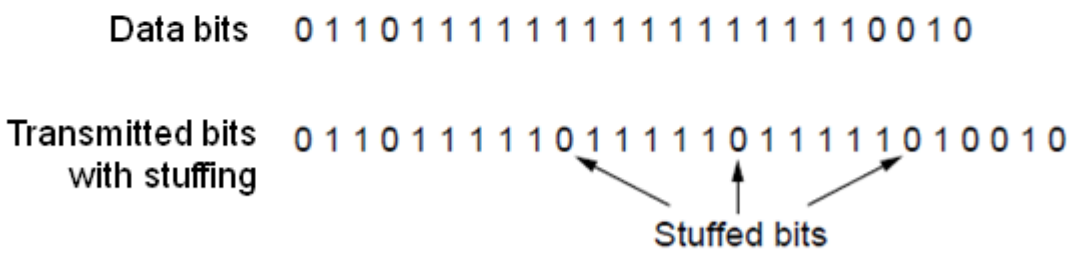
- Longer, but easy to resynchronize after error



Framing: Bit stuffing

Stuffing done at the bit level:

- Frame flag has six consecutive 1s (not shown)
- On transmit, after five 1s in the data, a 0 is added
- On receive, a 0 after five 1s is deleted



Framing

- Methods:
  - Byte count »
  - Flag bytes with byte stuffing »
  - Flag bits with bit stuffing »
  - Physical layer coding violations
    - »Use non-data symbol to indicate frame
- Why do we need framing?

CSE 461 – Coding & Framing at the Application Layer

(and Layering too)

John Zahorjan

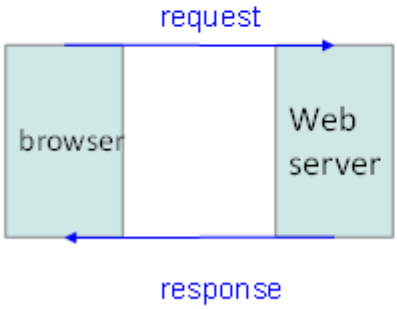
Spring 2012

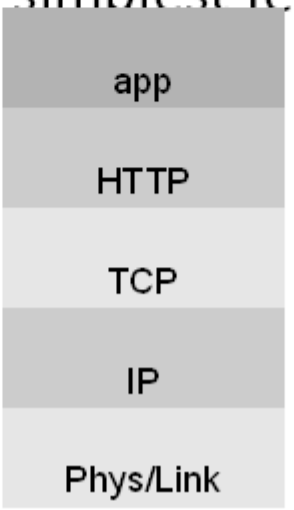
Example: HTTP

Want to fetch a web page in your browse

http://www.cs.uw.edu/

- This is done using an application layer protocol called *HTTP*
- Here we look at its simplest features





HTTP Messages

Request

GET /index.html HTTP/1.1<cr><lf>
Host: www.cs.washington.edu<cr><lf>
<cr><lf>

Response

HTTP/1.1 200 OK<cr><lf>
Date: Mon, 02 Apr 2012 08:54:55 GMT<cr><lf>
Server: Apache/2.2.22 (Fedora)<cr><lf>
Accept-Ranges: bytes<cr><lf>
Connection: close<cr><lf>
Content-Type: text/html<cr><lf>
Content-Language: en<cr><lf>
<cr><lf>
[web page payload]

Web Page Statistics

https://developers.google.com/speed/articles/web-metrics
May 2010

Metric	Top Sites	All Sites	Description
Pages	380 million	4.2 billion	Number of sample pages analyzed.
Resources	42.14	43.91	Average number of resources per page.
GETs	42.63	44.56	Average number of GETs per page. Similar to number of resources, but also includes redirects.
Hosts	8.39	7.01	Average number of unique hostnames encountered per page.
Resources Per Host	5.02	6.26	Average number of resources per host (derived from the 'Resources' and 'Hosts' values).
Network Size/KB	312.04	320.24	Average size transferred over the network per page, including HTTP headers. If resources were compressed, this would use the compressed size.
Document Size/KB	477.26	376.67	Average uncompressed size of a page and its resources, excluding HTTP headers.
Zippable Size/KB	287.51	170.16	Average uncompressed size of the compressible resources on a page, i.e., those with a Content-Type of 'text/*' or equivalent.

Google SPDY (HTTP 2.0)

http://tools.ietf.org/html/draft-mbelshe-httpbis-spdy-00 (Feb. 2012)

With some hand waving, web page render latency is
transmission time + propagation + server time + browser time

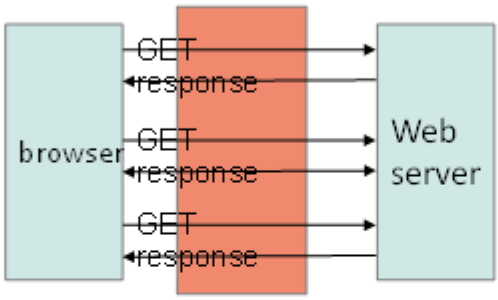
Tranmission time: compress the HTTP header (and, optionally, the payload)

Propagation delay: multiplex many requests over a single TCP connection

Bonus feature: Always encrypt

Stream Multiplexing

HTTP:



SPDY:

Performance Experiments

http://dev.chromium.org/spdy/spdy-whitepaper (2009?)

Table 1: Average page load times for top 25 websites

	DSL 2 Mbps downlink, 375 kbps uplink		Cable 4 Mbps downlink, 1 Mbps uplink	
	Average ms	**Speedup**	**Average ms**	**Speedup**
HTTP	3111.916		2348.188	
SPDY basic multi-domain* connection / TCP	2242.756	27.93%	1325.46	43.55%
SPDY basic single-domain* connection / TCP	1695.72	45.51%	933.836	60.23%
SPDY single-domain + server push / TCP	1671.28	46.29%	950.764	59.51%
SPDY single-domain + server hint / TCP	1608.928	48.30%	856.356	63.53%
SPDY basic single-domain / SSL	1899.744	38.95%	1099.444	53.18
SPDY single-domain + client prefetch / SSL	1781.864	42.74%	1047.308	55.40%

SPDY Protocol Stack

SPDY Request Header


```
+----------------------------------+
|1|    version    |         1      |
+----------------------------------+
|  Flags (8)  |  Length (24 bits)  |
+----------------------------------+
|X|           Stream-ID (31bits)   |
+----------------------------------+
|X| Associated-To-Stream-ID (31bits) |
+----------------------------------+
| Pri|Unused | Slot |              |
+-------------------+              |
| Number of Name/Value pairs (int32) |
+----------------------------------+
|     Length of name (int32)       |
+----------------------------------+
|           Name (string)          |
+----------------------------------+
|     Length of value  (int32)     |
+----------------------------------+
|          Value   (string)        |
+----------------------------------+
|           (repeats)              |
```

HTTP 2.0 / Microsoft

http://tools.ietf.org/html/draft-montenegro-httpbis-speed-mobility-01

Network Working Group
Internet-Draft
Expires: September 2, 2012

R. Trace
A. Foresti
S. Singhal
O. Mazahir
H. Nielsen
B. Raymor
R. Rao
G. Montenegro
Microsoft
Mar 2012

HTTP Speed+Mobility
draft-montenegro-httpbis-speed-mobility-01

Abstract

This document describes "HTTP Speed+Mobility," a proposal for HTTP 2.0 that emphasizes performance improvements and security while at the same time accounting for the important needs of mobile devices and applications. The proposal starts from both the Google SPDY protocol and the work the IETF has done around WebSockets. The proposal is not a final product but rather is intended to form a baseline for working group discussion.