University of Glasgow

Wrap-up

Networked Systems 3
Lecture 19

Lecture Outline

- Lessons from the laboratory exercise
- Future challenges
- Examination
- Networked systems in level 4

Lessons from the Laboratory Exercise

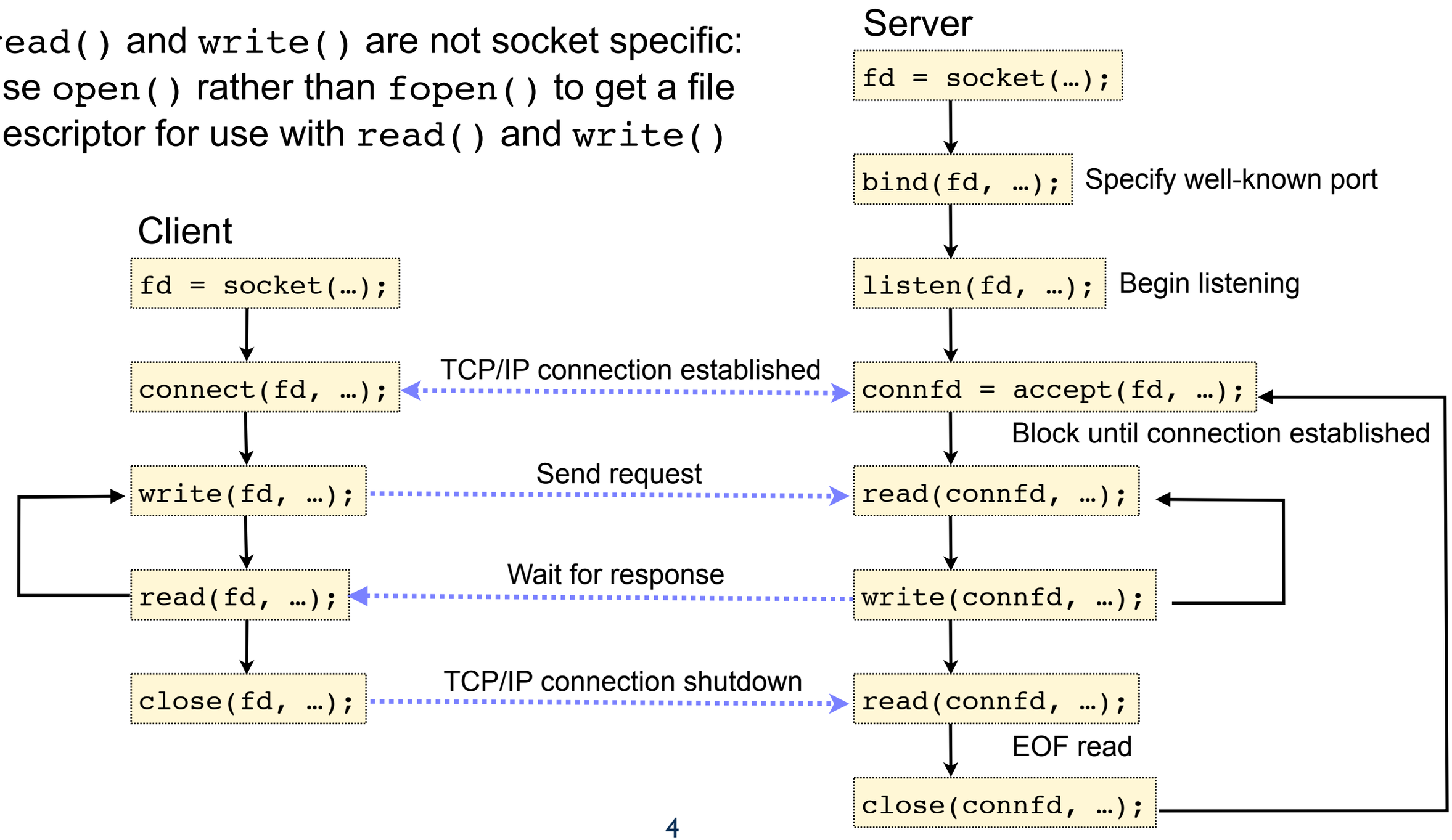
- Simple web client and server written in C
  - Using the Berkeley sockets API
  - Parsing requests
  - Generating responses
  - Threads and blocking IO

- High level issues:
  - Write a Makefile
  - Compile using `gcc -W -Wall` and *fix any warnings – they indicate likely problems in your code*
  - Use the `man` and `apropos` commands, don't look-up man pages online
    (you'll likely find the FreeBSD man pages – which are much better written than the Linux ones, but describe subtly different functions)

Using the Berkeley Sockets API

Remember to allocate sufficient memory

`read()` and `write()` are not socket specific: use `open()` rather than `fopen()` to get a file descriptor for use with `read()` and `write()`



Parsing Requests

- Incremental parsing of protocol headers difficult – whatever language you write it in
- String parsing in C surprisingly flexible:
  - `sscanf()` – pass in pointers; check the return code; behaviour of `%s`
  - `strchr()`
  - `strstr()`
  - `strtok()` – modifies string being tokenised; not thread safe: `strtok_r`
  - `strcmp()`
- But easy to leave buffer overflows, or use memory after `free()`
  - Be careful – often security vulnerabilities

Example: Parsing a URL

```
char empty_path[] = "";

static int
parse_http_url(char *url, char **hostname, char **filename)
{
    char *context;

    if ((url == NULL) || (strlen(url) < 7) || (strncmp(url, "http://", 7) != 0)) {
        return 1;
    }

    *hostname = strtok_r(url + 7, "/", &context);
    *filename = strtok_r(NULL,     "", &context);

    if (*hostname == NULL) {
        return 1;
    }

    if (*filename == NULL) {
        *filename = empty_path;
    }

    return 0;
}
```

Pointer arithmetic

Original URL modified, to null terminate hostname

Generating Responses

```
char *buf = malloc(16 + strlen(filename));

strcat(buf, “GET “);
strcat(buf, filename);
strcat(buf, “ HTTP/1.1\r\n”);
```



```
char *buf = malloc(16 + strlen(filename));

sprintf(buf, “GET %s HTTP/1.1\r\n”, filename);
```

Be careful to allocate enough memory

Use `sprintf()` or `strncat()` in preference to `strcat()`

Beware buffer overflows with `strcat()`

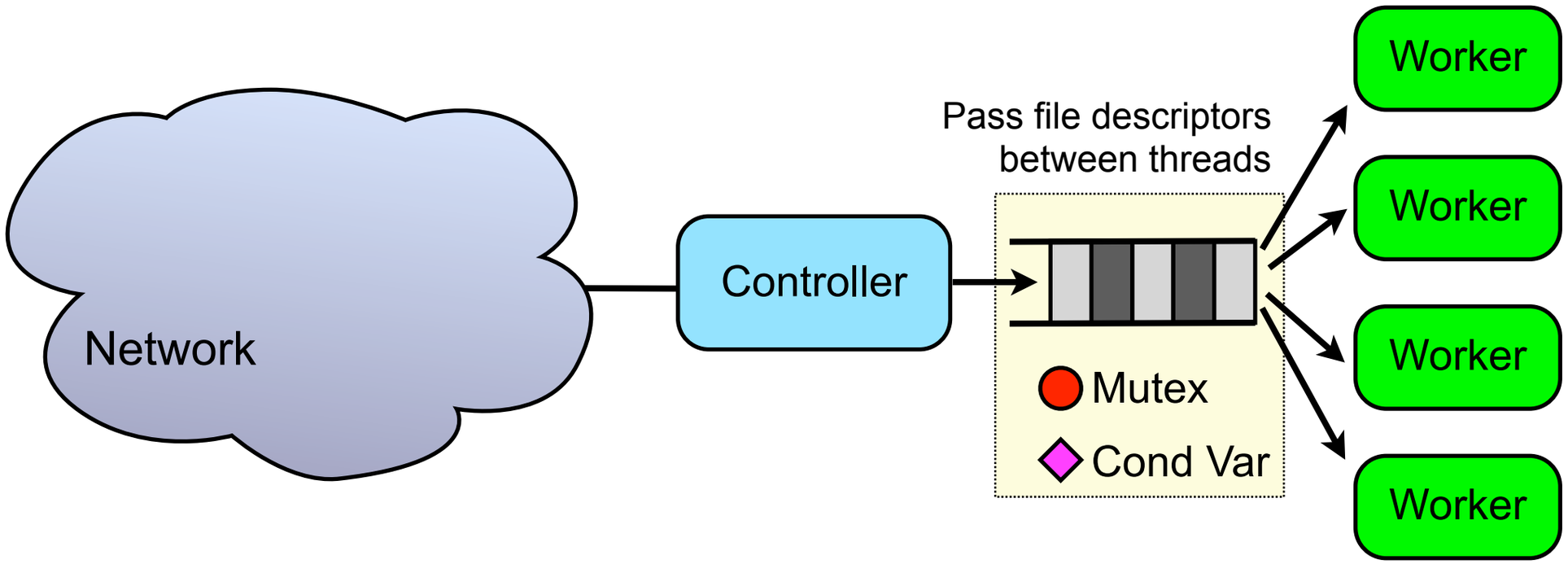
Write data to a TCP socket in large chunks where possible (helps the congestion control algorithm get good performance)

Beware the difference between strings and binary data: `strcpy()` *vs.* `memcpy()` and `memmove()`

Threads and Blocking IO

- Problem: how to process data from multiple file descriptors?
  - Use blocking `read()` calls with a thread pool – pthreads
  - Use `select()` and process all events in a single thread

Thread Pool



- Controller thread – accepts connections, adds file descriptors to work queue
- Pool of worker threads – take file descriptors from queue, `read()` data, generate and `write()` responses
- Access to work queue protected by a mutex; workers wait on a condition variable

Managing connections with `select()`

```
#include <sys/select.h>
...
int             fd1, fd2, fd3;
...
int             s;
int             nfds = max(fd1, fd2, fd3) + 1;
fd_set          read_fds;
struct timeval  timeout;

timeout.tv_secs  = 0;
timeout.tv_usecs = 10000; // 10ms


FD_ZERO(&rfds);
FD_SET(fd1, &rfds);
FD_SET(fd2, &rfds);
FD_SET(fd3, &rfds);

s = select(nfds, &rfds, NULL, NULL, &timeout);
if (s  < 0) ... // Error
if (s == 0) ... // Timeout
if (s  > 0) {
    if (FD_ISSET(fd1, &rfds)) ...
    if (FD_ISSET(fd2, &rfds)) ...
    if (FD_ISSET(fd3, &rfds)) ...
}
```

Maximum value of the file descriptors, plus 1

Readable?

Have errors?

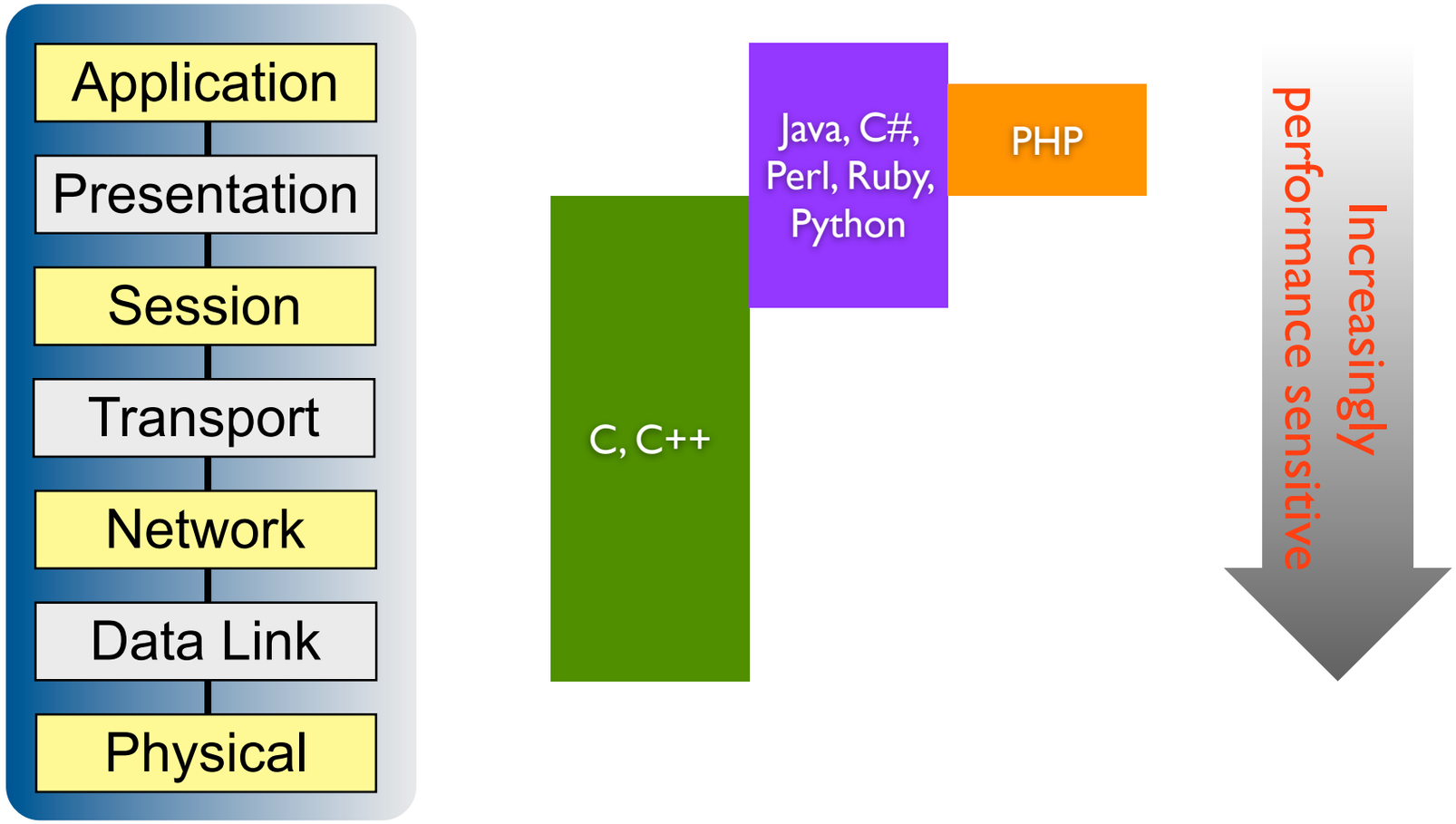
Can be written?

Future Challenges

- We have the Internet and the World-Wide Web… is networking done?

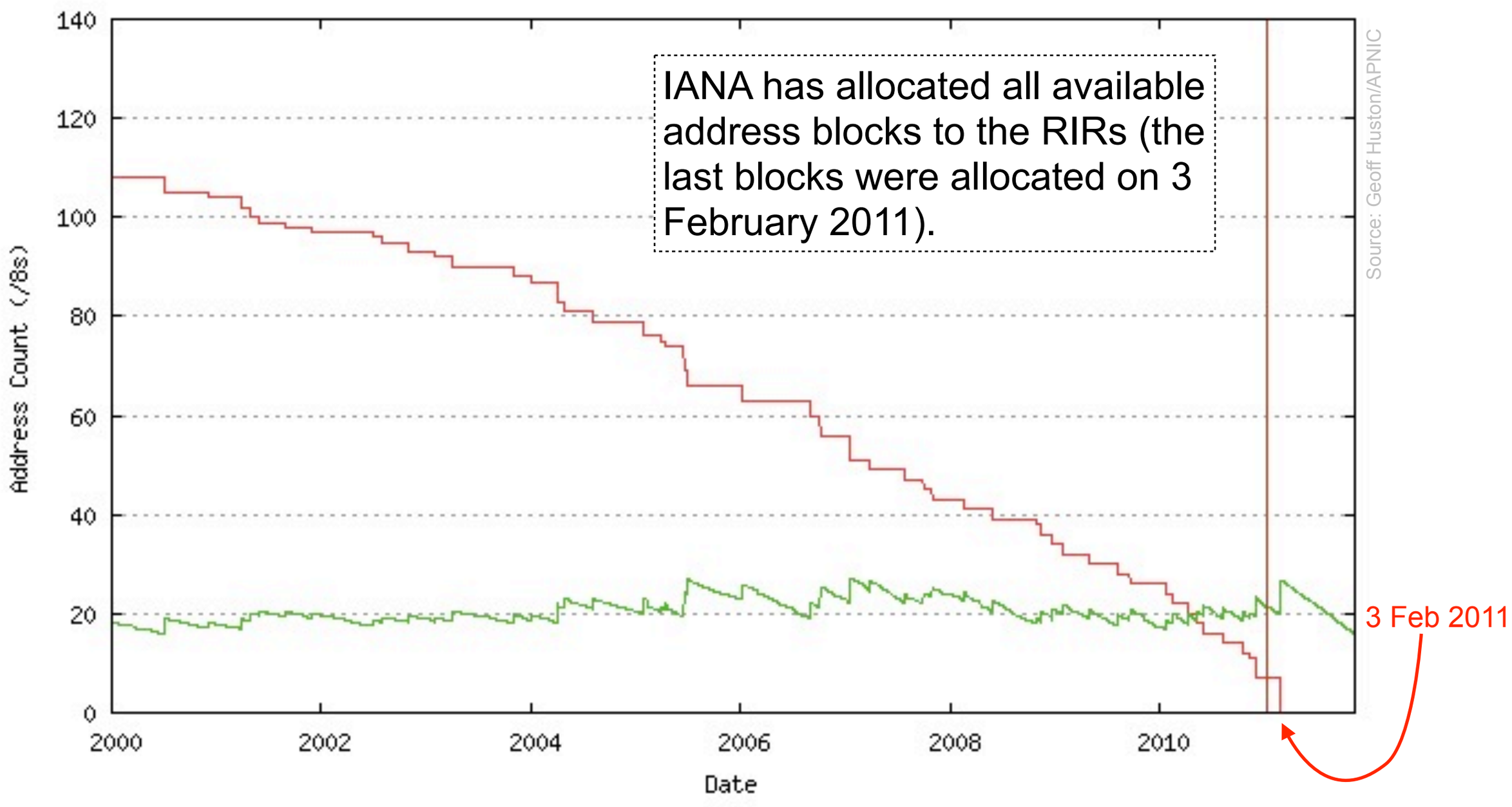
- No!

Future Challenges: Implementation



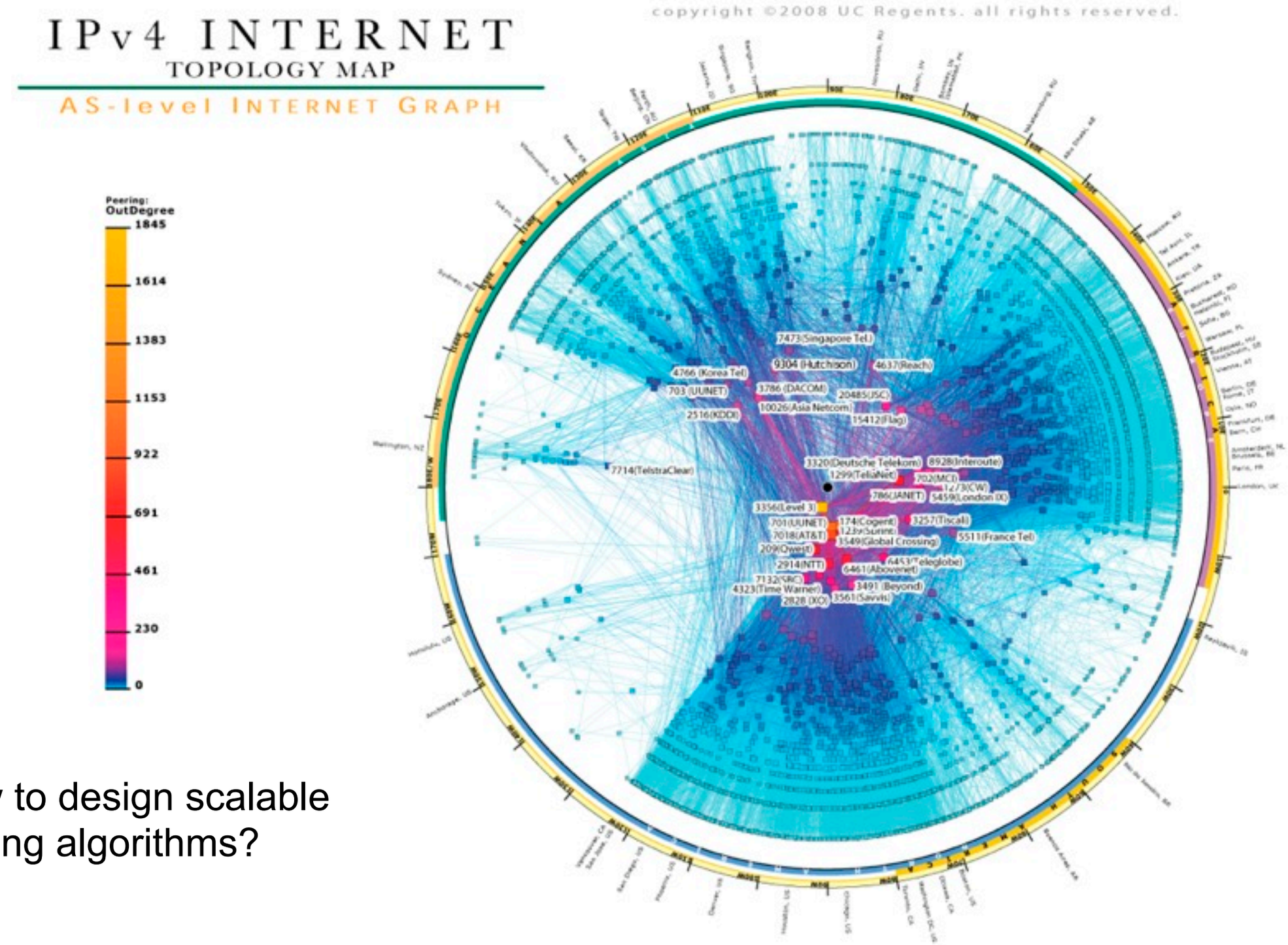
Research issue: to what extent can the use of safe languages be pushed down to the lower layers of the protocol stack?

Future Challenges: Addressing and IPv6



Can we transition to IPv6? How do we deal with the problems inherent in multi-NAT if the transition fails?

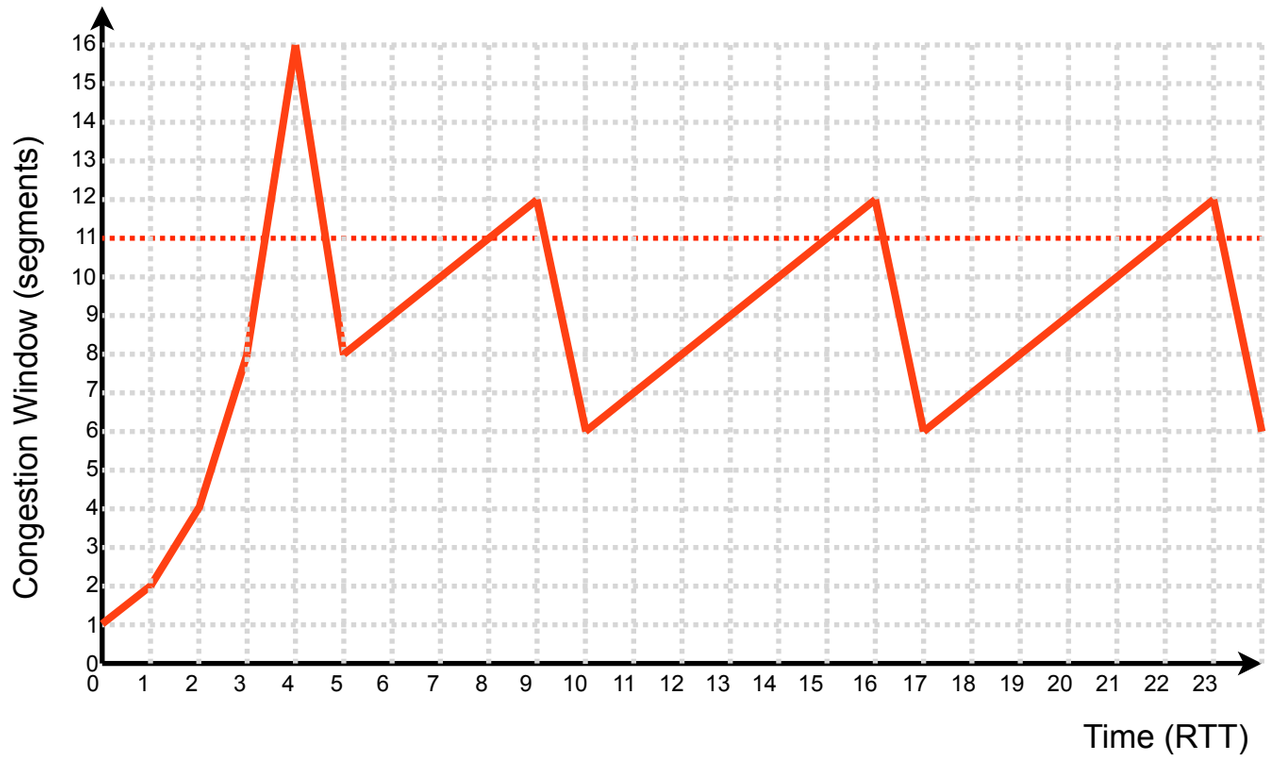
Future Challenges: Routing



How to design scalable routing algorithms?

Source: CAIDA

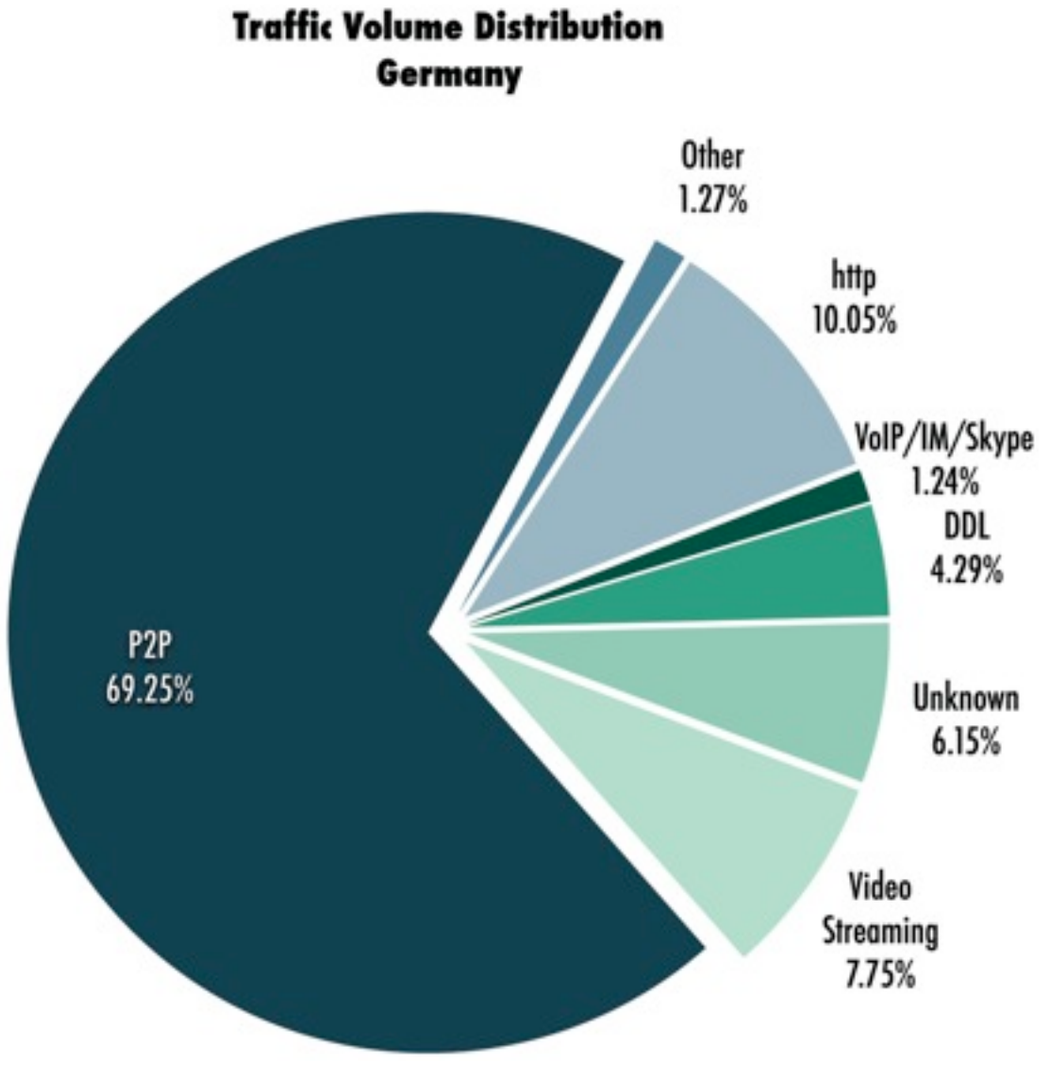
Future Challenges: Congestion Control



How to share capacity between TCP-based and real-time media?

Future Challenges: Peer-to-Peer Systems

- How to tame peer-to-peer applications?
  - How to limit scope of traffic, to avoid overwhelming the network?
  - How to prevent illegal distribution of content/how to avoid censors?



Source: ipoque Internet Study, 2007

Future Challenges: Wireless & MANET



Source: Tegola project, University of Edinburgh

Networked Systems in Level 4

- Two taught modules cover networked systems:
  - Advanced Networking and Communications 4
  - Distributed Algorithms and Systems 4
- Individual projects in networked systems:
  - Look for projects supervised by members of the Embedded, Networked, and Distributed Systems research group (happy to consider self-defined projects)
    - Joe Sventek
    - Colin Perkins (http://csperkins.org/research/)
      - High-level languages for low-level protocols
      - Next generation Internet routing algorithms; visualisation
      - Adaptive IPTV streaming
    - Dimitrios Pezaros
    - Lewis Mackenzie

Examination

- Worth 80% of course mark
- Format of exam:
  - 1.5 hours
  - Answer all 3 questions
- Copies of past exam papers are on Moodle

The End