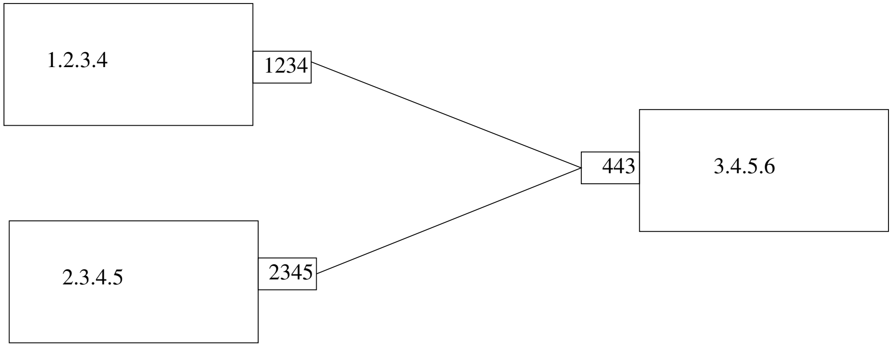
Sockets

Client-server architectures

- Have *server* process, which waits for client requests and processes them once they arrive
- Multiple clients may connect and request service
- Standard paradigm for services on the internet
- Each computer which is connected to the internet has a unique IP address `xxx.xxx.xxx.xxx`, where `xxx` is a number between 0 and 255
- This IP address can be used for data exchange with this computer from anywhere on the internet
- Have in addition to IP-addresses *port numbers* which are numbers which identify endpoints of a connection on a computer

- Each connection is identified by the source IP address and source port, and the destination IP address and destination port.
- Have specified destination ports assigned to particular services
- Examples:
  - Port 80 for http
  - Port 25 for sending mail (smtp)
  - Port 143 for reading mail via imap
- Port assignments listed in `/etc/services`


1.2.3.4
1234
2.3.4.5
2345
443
3.4.5.6

Sockets

Sockets are the endpoints for a client-server connection
Setup for the server:

- Server creates endpoint via `socket`-system call
- Server specifies port number and protocol in structure `sockaddr_in6`
- Server assigns information in `sockaddr_in6`-structure to socket via `bind`-system call

Now server waits for incoming connection via `accept`-system call
When connection received, server reads data from client via `read`-system call and writes data back to server via `write`-system call
When server is finished with current connection, server closes connection via `close`-system call

Sockets, continued

Setup for the client:

- Client creates endpoint via `socket`-system call
- Client connects to server via `connect`-system call
- Client write data to server via `write`-system call and reads data from server via `read`-system call
- When client is finished with current connection, client closes connection via `close`-system call

Concurrency and Sockets

- Good handling of concurrency vital for implementing sockets
- Key point: Server program may create separate thread for each incoming connection arbitrary number of threads, which share memory and may run concurrently
- Gives rise to possible race conditions which require synchronisation
- Use suitable mutual exclusion, as previously discussed