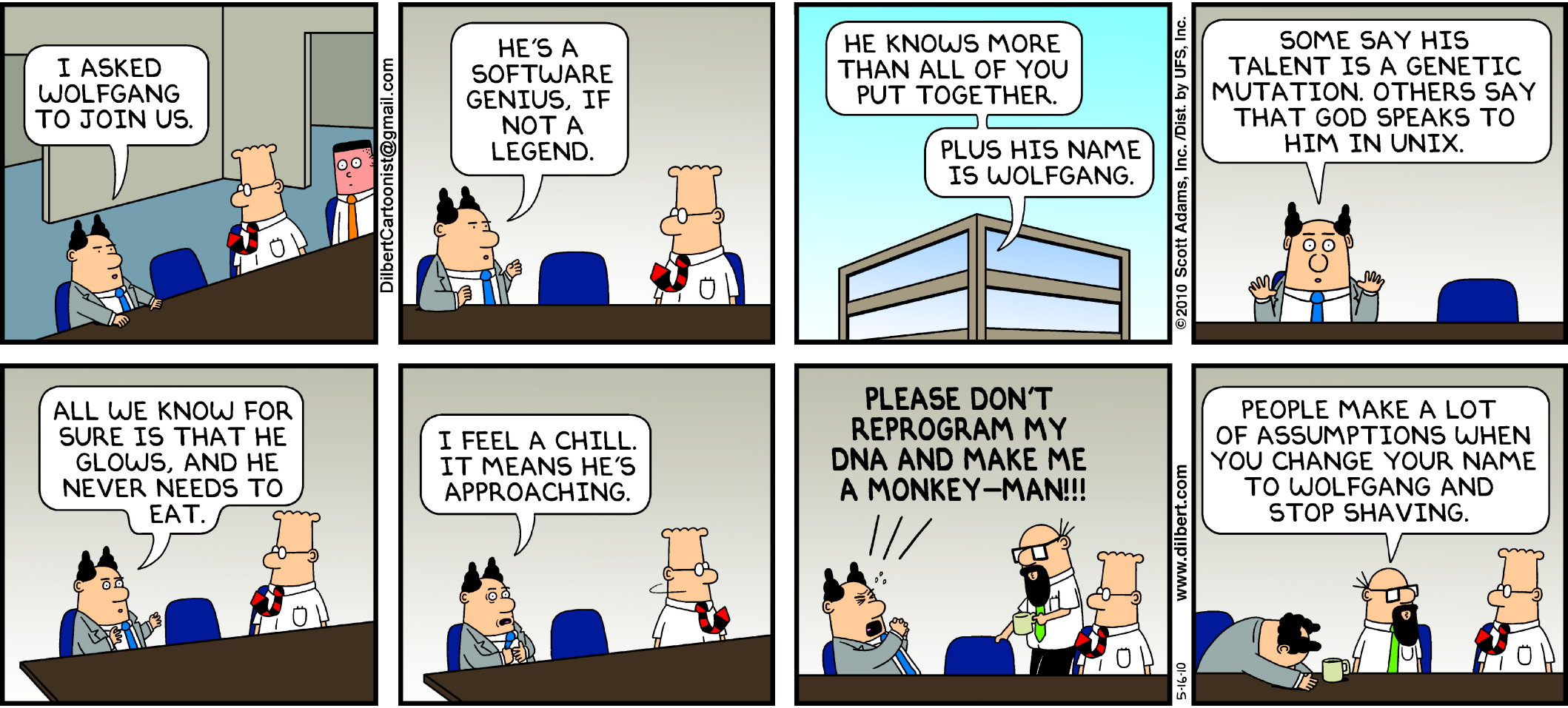
Life, Liberty, and the Pursuit of...Sockets!

Athula Balachandran
IIT-M, GHC 7609

Wolf Richter
UVA, GHC 9127

Willkommen bei 15-441



15-441 ലേക്ക് സ്വാഗതം

Who took 15-213?

And built an HTTP 1.0 proxy?

Okay, who knows Python?

Oh Really?

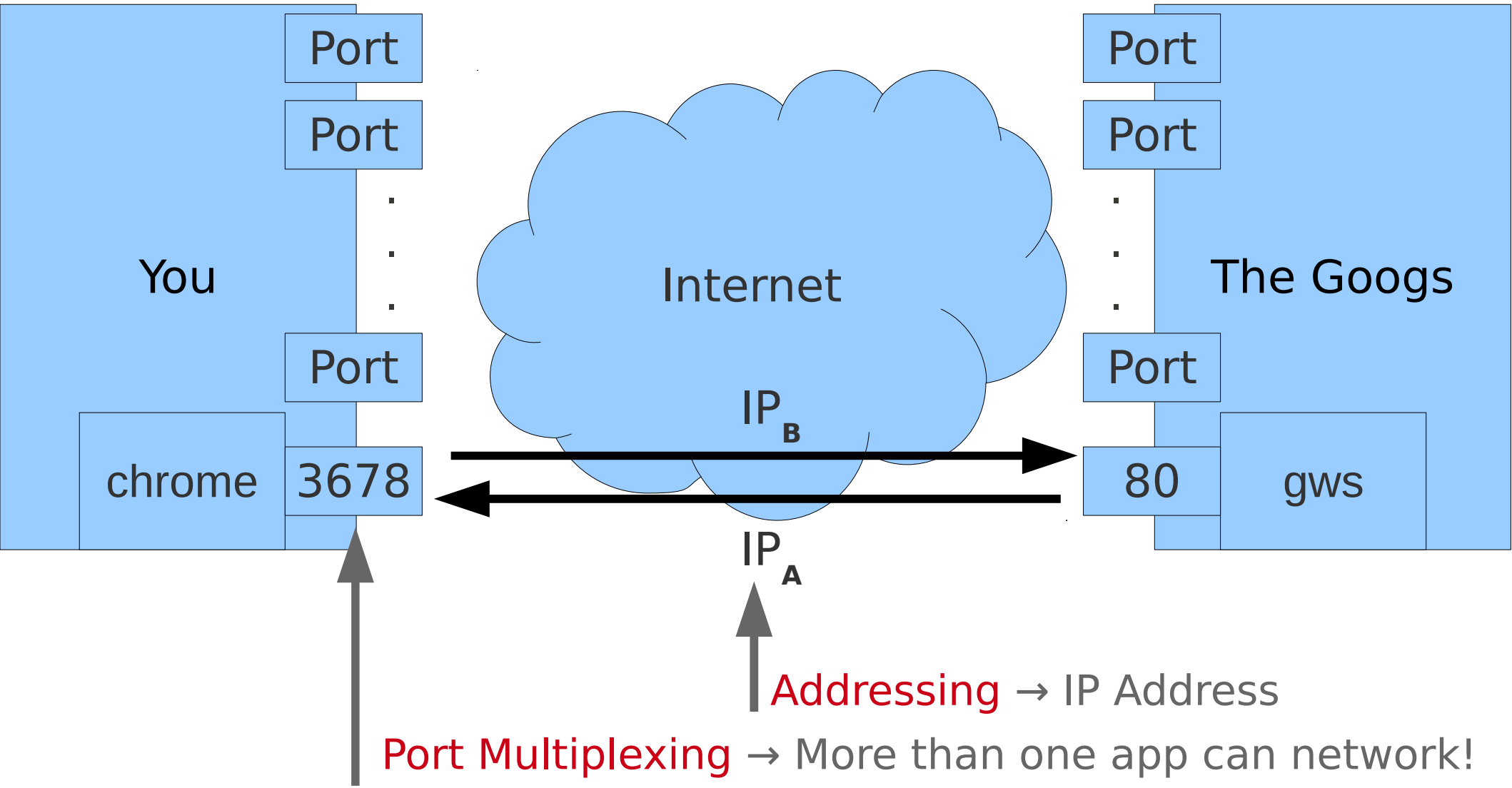
```
----------------Python Code-----------------------
def fun1():
t = 0
f = lambda x: t
t = 1
return f

def fun2():
t = 0
f = (lambda y: lambda x: y) (t)
t = 1
return f

v1 = fun1()(0)
v2 = fun2()(0)
print v1, v2
--------------------------------------------------
```

Q: What are v1 and v2?

What's in a socket?

- I want computer A to talk to computer B?
- But how:
  - Request-Reply: Synchronous, RPC
  - Publish-Subscribe: Asynchronous, Broadcast
  - Push-Pull: Worker queue, Round Robin
- Source (IP/TCP) <ip,port>
- Destination (IP/TCP) <ip,port>

IP? Port!?



Addressing → IP Address

Port Multiplexing → More than one app can network!

CS: Acronyms Galore!

- The Internet Assigned Numbers Authority (IANA)

  http://www.iana.org/

- The Internet Engineering Task Force (IETF)

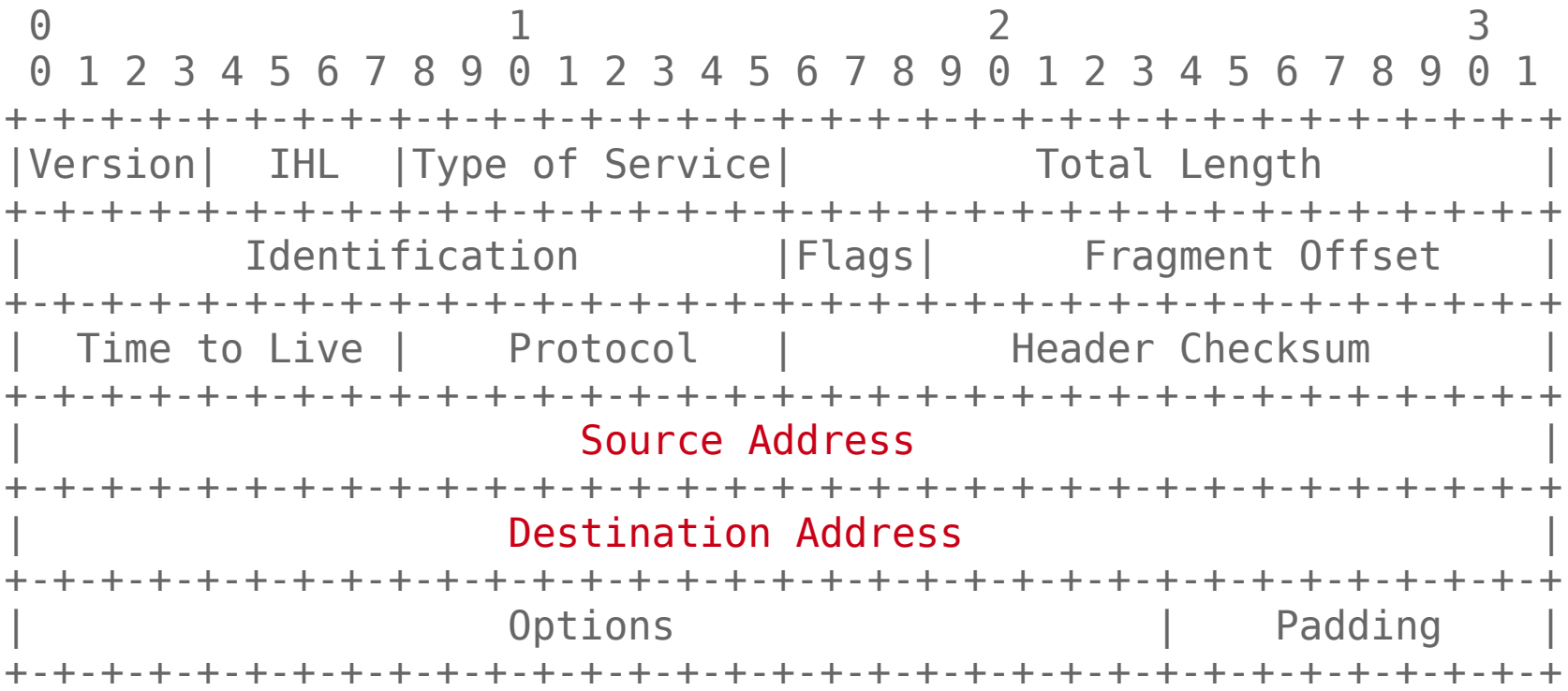
  http://www.ietf.org

- Internet Corporation for Assigned Names and Numbers (ICANN)

  http://www.icann.org/

IPv4 Details: RFC791

http://www.ietf.org/rfc/rfc791.txt

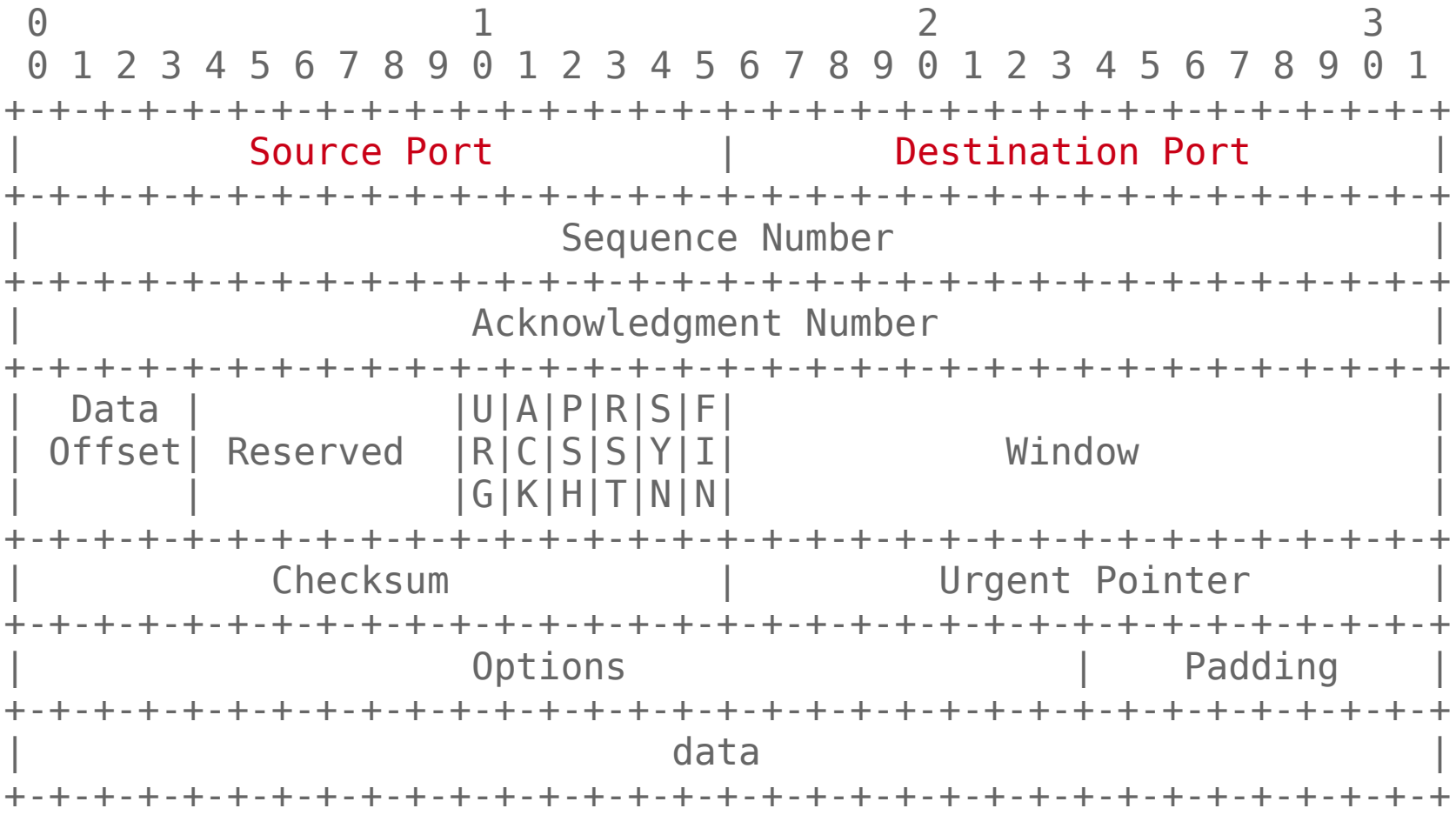


Example Internet Datagram Header

Figure 4.

TCP Details: RFC793 + Others

http://www.ietf.org/rfc/rfc793.txt



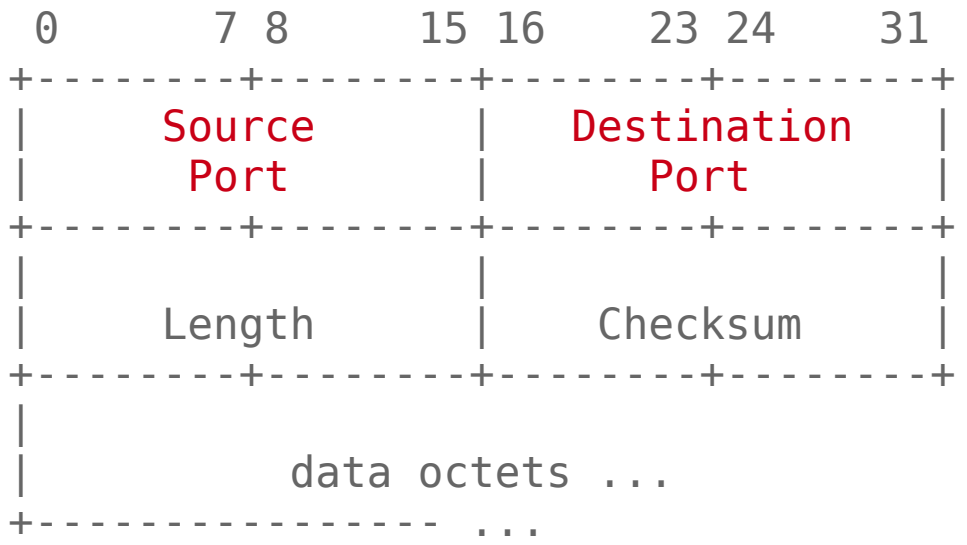
TCP Header Format

Note that one tick mark represents one bit position.

Figure 3.

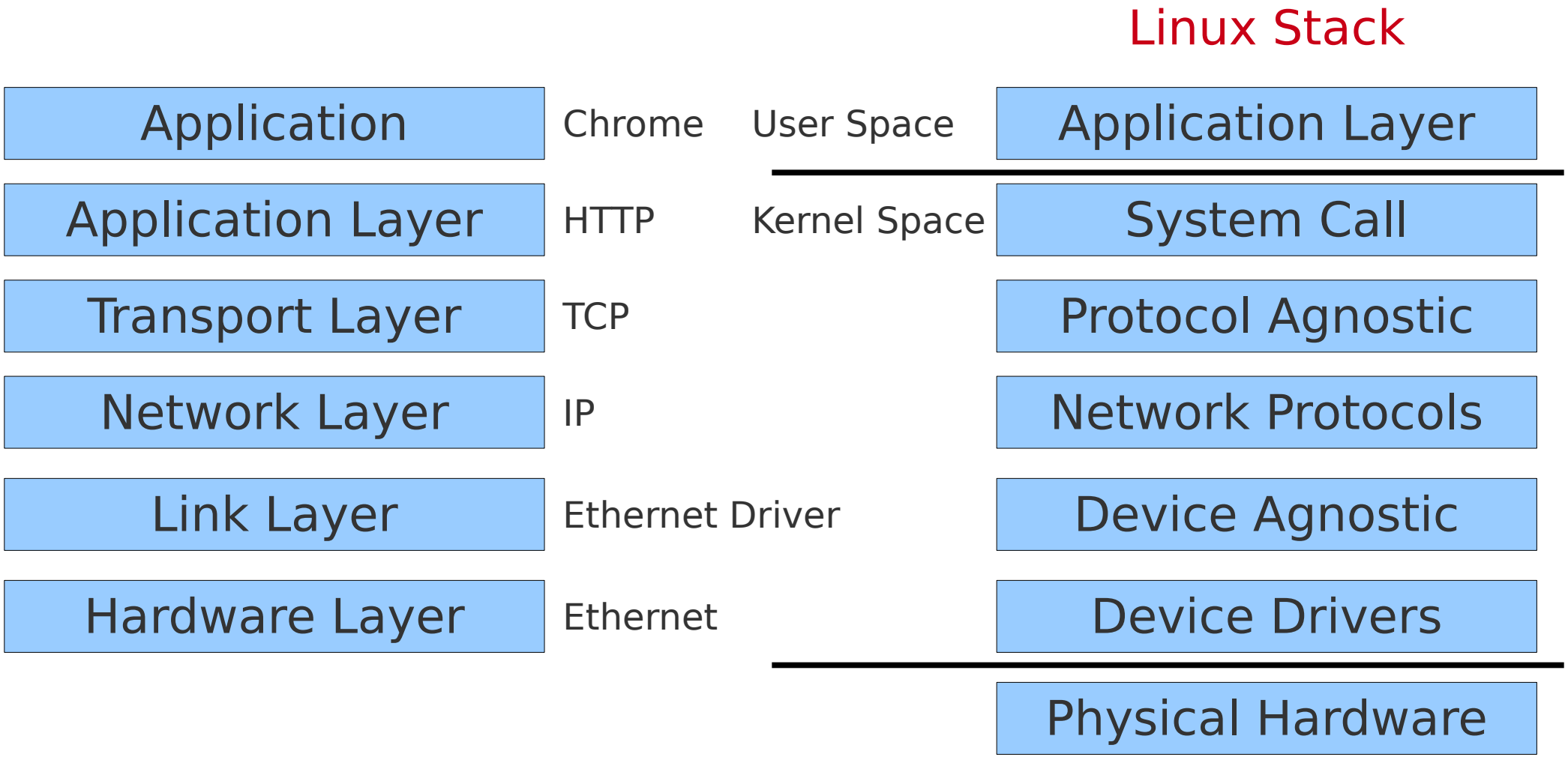
UDP Details: RFC768

http://www.ietf.org/rfc/rfc768.txt



User Datagram Header Format

The Network Stack



Network Communications

- 32 bit—4,294,967,296 possible addresses
- 16 bit—65,536 possible ports [0,65535]
- < 1024 reserved ports on most Unices
  - Your new app should use >1024
- Some common ports:
  - 80 – Hypertext Transport Protocol (HTTP)
  - 22 – Secure Shell (SSH)
  - 25 – Simple Mail Transfer Protocol (SMTP)

How to: Server

- Create a socket via `socket()`
- Bind to an endpoint via `bind()`
- Listen for connections via `listen()`
- Accept connections via `accept()`
- Read from socket via `recv()`
- Write to socket via `send()`

socket()

```
#include <sys/socket.h>
int socket(int socket_family, int socket_type, int protocol);
```

Generally set `socket_family` to PF_INET → IPv4

`socket_type` to SOCK_STREAM → TCP

`protocol` to 0 → Yes, TCP

socket() Code Example

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h>

int main(int argc, char* argv[])
{
    int sock = socket(PF_INET, SOCK_STREAM, 0);
    return EXIT_SUCCESS;
}
```

socket_family

Family	Meaning
PF_UNIX, PF_LOCAL	Local communication
PF_INET	IPv4
PF_INET6	IPv6
PF_IPX	IPX - Novell
PF_NETLINK	Kernel user interface device
PF_X25	ITU-T X.25
PF_AX25	Amateur radio AX.25
PF_ATMPVC	Raw ATM PVCs
PF_APPLETALK	Appletalk
PF_PACKET	Low level packet interface

socket_type

Type	Meaning
SOCK_STREAM	Sequenced, reliable TCP
SOCK_DGRAM	Connectionless UDP
SOCK_SEQPACKET	SOCK_STREAM w/ fixed length
SOCK_RAW	Raw network protocol access
SOCK_RDM	Reliable datagram no order
SOCK_PACKET	Deprecated

`socket()` Error States

Type	Meaning
EACCESS	Permission denied
EAFNOSUPPORT	Address family unsupported
EINVAL	Unknown protocol
EMFILE	Process file table overflow
ENFILE	Total open files limit reached
ENOBUFS or ENOMEM	System out of memory
EPROTONOSUPPORT	Specified protocol not supported

bind()

```
#include <sys/socket.h>

int bind(int sock, const struct sockaddr* addr, socklen_t addrlen);
```

Generally create a socket as described, then create a sockaddr struct.

bind() Code Example

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <netinet/in.h>

int main(int argc, char* argv[])
{
    int sock = socket(PF_INET, SOCK_STREAM, 0);
    struct sockaddr_in addr;
    addr.sin_family = AF_INET;
    addr.sin_port = htons(1025);
    addr.sin_addr.s_addr = INADDR_ANY;
    int err = bind(sock, (struct sockaddr *)
                   &addr, sizeof(addr));
    return EXIT_SUCCESS;
}
```

sin_family

Address Family	Meaning
AF_INET	IPv4
AF_INET6	IPv6
AF_UNIX	Local communication
AF_APPLETALK	Appletalk addressing
AF_PACKET	Packet addressing
AF_X25	ITU-T X.25 addressing
AF_NETLINK	Between kernel and user

sin_port

- Set to port you want to connect/listen

Conversion Function	Explanation
`uin16_t htons(uint16_t)`	Host to network short
`uin32_t htonl(uint32_t)`	Host to network long
`uin16_t ntohs(uint16_t)`	Network to host short
`uin32_t htonl(uint32_t)`	Network to host long

`bind()` Error States

Type	Meaning
EADDRINUSE	Address in use already
EBADF	Invalid socket
EINVAL	Socket already bound
ENOTSOCK	File descriptor, not socket

listen()

```
#include <sys/socket.h>
int listen(int sock, int backlog);
```

After `bind()`ing, you can `listen()` for connections.

listen() Code Example

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <netinet/in.h>

int main(int argc, char* argv[])
{
    int sock = socket(PF_INET, SOCK_STREAM, 0);
    struct sockaddr_in addr;
    addr.sin_family = AF_INET;
    addr.sin_port = htons(1025);
    addr.sin_addr.s_addr = INADDR_ANY;
    int err = bind(sock, (struct sockaddr *)
                  &addr, sizeof(addr));
    int err2 = listen(sock, 5);
    return EXIT_SUCCESS;
}
```

backlog

- Queue length for incoming sockets
- Fully established already

`listen()` Error States

Type	Meaning
EADDRINUSE	Port already listened to
EBADF	Not a valid descriptor
ENOTSOCK	Not a socket passed in
EOPNOTSUPP	Can't `listen()` on this socket

accept()

```
#include <sys/socket.h>

int accept(int sockfd, struct sockaddr * addr, struct socklen_t * len);
```

After `listen()`ing, you can `accept()` connections.

Pass in the socket from before, and pointers to data structures defined by you.

These represent connected client state for future use.

accept() Code Example

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <netinet/in.h>

int main(int argc, char* argv[])
{
    int sock = socket(PF_INET, SOCK_STREAM, 0);
    struct sockaddr_in addr, caddr;
    addr.sin_family = AF_INET;
    addr.sin_port = htons(1025);
    addr.sin_addr.s_addr = INADDR_ANY;
    int err = bind(sock, (struct sockaddr *)
                &addr, sizeof(addr));
    int err2 = listen(sock, 5);
    socklen_t len = sizeof(caddr);
    int client = accept(sock, (struct sockaddr *)
                        &caddr, &len);
    return EXIT_SUCCESS;
}
```

`accept()` Error States 1

Type	Meaning
EAGAIN or EWOULDBLOCK	No connections, non-blocking
EBADF	Not a valid descriptor
ECONNABORTED	Connection was aborted
EINTER	Interrupted before connection
EINVAL	Socket not listening
EMFILE	Per-process open file limit reached
ENFILE	System open file limit reached

`accept()` Error States 2

Type	Meaning
ENOTSOCK	Descriptor of file, not socket
EOPNOTSUPP	Socket is not SOCK_STREAM
EFAULT	Can not write to addr
ENOBUFS, ENOMEM	Out of memory
EPROTO	Protocol error
EPERM	Firewall rules forbid connection

recv()

```
#include <sys/socket.h>
int recv(int sockfd, void * buf, size_t len, int flags);
```

After `accept()`ing, you can `recv()` data.

For now set `flags` to 0

recv() Code Example

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <netinet/in.h>

int main(int argc, char* argv[])
{
    char buf[256];
    int sock = socket(PF_INET, SOCK_STREAM, 0);
    struct sockaddr_in addr, caddr;
    addr.sin_family = AF_INET;
    addr.sin_port = htons(1025);
    addr.sin_addr.s_addr = INADDR_ANY;
    int err = bind(sock, (struct sockaddr *)
              &addr, sizeof(addr));
    int err2 = listen(sock, 5);
    socklen_t len = sizeof(caddr);
    int client = accept(sock, (struct sockaddr *)
                        &caddr, &len);
    ssize_t read = recv(client, buf, 256, 0);
    return EXIT_SUCCESS;
}
```

flags

Flag	Meaning
MSG_DONTWAIT	Non-blocking check for data
MSG_ERRQUEUE	Receive errors from socket
MSG_OOB	Out-of-band data request
MSG_PEEK	Return data without removing
MSG_TRUNC	Return real packet length
MSG_WAITALL	Block until full request satisfied

recv() Error States 1

Type	Meaning
EAGAIN	Would block but not supposed
EBADF	Not a valid descriptor
ECONNREFUSED	Remote host refused conn
EFAULT	Receive buffer pointer bad
EINTER	Receive interrupted by interrupt
EINVAL	Invalid argument passed
ENOMEM	Out of Memory
ENOTCONN	Socket not connected yet
ENOTSOCK	Socket argument isn't a socket

send()

```
#include <sys/socket.h>
int send(int sockfd, void * buf, size_t len, int flags);
```

After `accept()`ing, you can `send()` data.

For now set `flags` to 0

send() Code Example

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <netinet/in.h>

int main(int argc, char* argv[])
{
    char buf[256];
    int sock = socket(PF_INET, SOCK_STREAM, 0);
    struct sockaddr_in addr, caddr;
    addr.sin_family = AF_INET;
    addr.sin_port = htons(1025);
    addr.sin_addr.s_addr = INADDR_ANY;
    int err = bind(sock, (struct sockaddr *)
              &addr, sizeof(addr));
    int err2 = listen(sock, 5);
    socklen_t len = sizeof(caddr);
    int client = accept(sock, (struct sockaddr *)
                        &caddr, &len);
    ssize_t sent = send(client, buf, 256, 0);
    return EXIT_SUCCESS;
}
```

flags

Flag	Meaning
MSG_CONFIRM	Link layer forward progress
MSG_DONTROUTE	Don't use a gateway
MSG_DONTWAIT	Non-blocking send
MSG_EOR	Terminate record
MSG_MORE	Caller has more data to send
MSG_NOSIGNAL	No SIGPIPE on error
MSG_OOB	Send out-of-band data

send() Error States 1

Type	Meaning
EACCESS	Write permission denied
EAGAIN or EWOULDBLOCK	Block not supposed to
EBADF	Invalid descriptor specified
ECONNRESET	Connection reset by peer
EDESTADDRREQ	Not connecting, no peer set
EFAULT	Invalid user address specified
EINTER	Signal interrupt
EINVAL	Invalid argument
EISCONN	Connected already

send() Error States 2

Type	Meaning
EMSGSIZE	Atomic send, too big message
ENOBUFS	Output queue physically full
ENOMEM	Out of memory
ENOTCONN	Socket not connected
ENOTSOCK	Argument is not a socket
EOPNOTSUPP	Some flags are incorrect
EPIPE	Local end shutdown

How to: Client

- Create a socket via `socket()`
- Connect to an endpoint via `connect()`
- Read from socket via `recv()`
- Write to socket via `send()`

connect()

```
#include <sys/socket.h>

int connect(int socket, const struct sockaddr *serv_addr, socklen_t protocol);
```

Use socket as before, get `serv_addr` from `getaddrinfo()`.

Free with `freeaddrinfo()`.

connect() Code Example

```
#include <netdb.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>

int main(int argc, char* argv[])
{
   struct addrinfo addr, *caddr;
   memset(&addr, 0, sizeof(addr));
   addr.ai_family = AF_UNSPEC;
   addr.ai_socktype = SOCK_STREAM;
   getaddrinfo("www.google.com", "80", &addr, &caddr);
   int sock = socket(caddr->ai_family, caddr->ai_socktype,
                     caddr->ai_protocol);
   connect(sock, caddr->ai_addr, caddr->ai_addrlen);
   return EXIT_SUCCESS;
}
```

ai_family

Family	Meaning
AF_UNSPEC	Any protocol family
AF_INET	IPv4
AF_INET6	IPv6

connect() Error States 1

Type	Meaning
EACCESS	Permission denied
EACCESS, EPERM	Connect broadcast without flag
EADDRINUSE	Local address already in use
EAFNOSUPPORT	Incorrect address family
EAGAIN	No more free local ports
EALREADY	Non-blocking, previous incomplete connection
EBADF	File descriptor not valid

`connect()` Error States 2

Type	Meaning
ECONNREFUSED	No remote listening socket
EFAULT	Socket address structure outside user address space
EINPROGRESS	Non-blocking, cannot immediately connect
EINTER	Interrupt by caught signal
EISCONN	Socket already connected
ENETUNREACH	Unreachable network
ENOTSOCK	File descriptor not a socket

Socket Programming Gotchas

- Endianness Matters: Network Byte Order
  - `htons()` - host to network short
  - `ntohs()` -network to host short
- Cleanup state—avoid memory leaks
  - `freeaddrinfo()`
  - Check correctness with `valgrind`
- Error Handling
  - Tedious, but worth it (and required!)
- Timeouts
  - Implement for robust networking behavior

Socket Programming Gotchas

- Never expect to `recv()` what you `send()`
  - Assume partial receipt of data possible
  - Use buffers intelligently to mitigate this
  - Send byte counts first, read until finished
- Prepare your code for random failures
  - We introduce random faults when grading
  - Test too—`ctrl+c` server and client randomly
- Cleanup Allocated Memory
  - `close()` sockets, etc.

Concurrency: `select()`

- Watch a set of sockets
- Use `select()` to find sockets ready for IO
- Server-side only—clients are agnostic

select()

```
#include <sys/select.h>
int select(int nfds, fd_set *readfds, fd_set *writefds,
           fd_set *exceptfds, struct timeval *timeout);
void FD_CLR(int fd, fd_set *set);
int FD_ISSET(int fd, fd_set *set);
void FD_SET(int fd, fd_set *set);
void FD_ZERO(fd_set *set);
```

After `accept()`ing, you timeout poll using `select()`.

Manipulate set of descriptors with `FD_*`, then `select()`.

Future recitation for more in-depth view.

Error Checking

- Read documentation first
- Sometimes you need to:

```
#include <errno.h>

...
switch (errno)
{

...
}
```

System Call Documentation:

POSIX – Portable Operating System Interface for Unix
IEEE 1003.1-2008, The Open Group

"POSIX.1-2008 defines a standard operating system interface and environment, including a command interpreter (or "shell"), and common utility programs to support applications portability at the source code level."

http://pubs.opengroup.org/onlinepubs/9699919799/

Also, more correct, your system's man pages!

Another excellent resource:

Beej's Guide to Network Programming

http://beej.us/guide/bgnet/

Project 1: HTTP déjà vu

- Blast from the past 15-213
- This time a real HTTP server with:
  - SSL
  - `select()` IO for concurrent connections
  - HTTP 1.1
  - Maybe an embedded Python wsgi interface
- Big project, **start early**!

Who ya gonna call?

The TAs!

abalacha@cs.cmu.edu
wolf@cs.cmu.edu

GitHub:

Find source for this presentation.

Find a full echo server and echo client.

Be cool.

git clone git://github.com/theonewolf/15-441-Recitation-Sessions.git