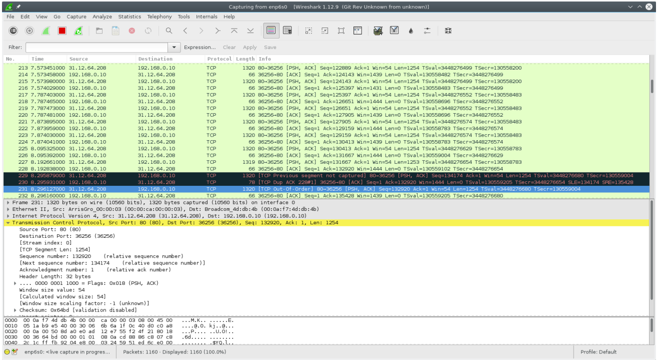
Networks and Network Programming

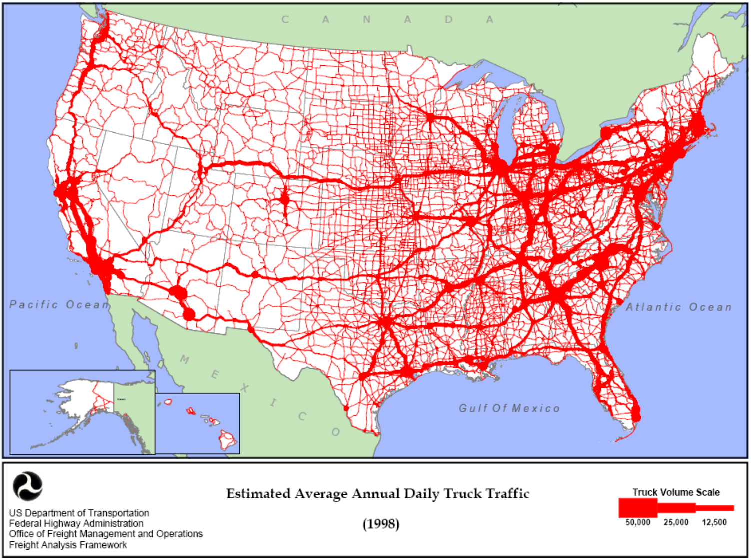


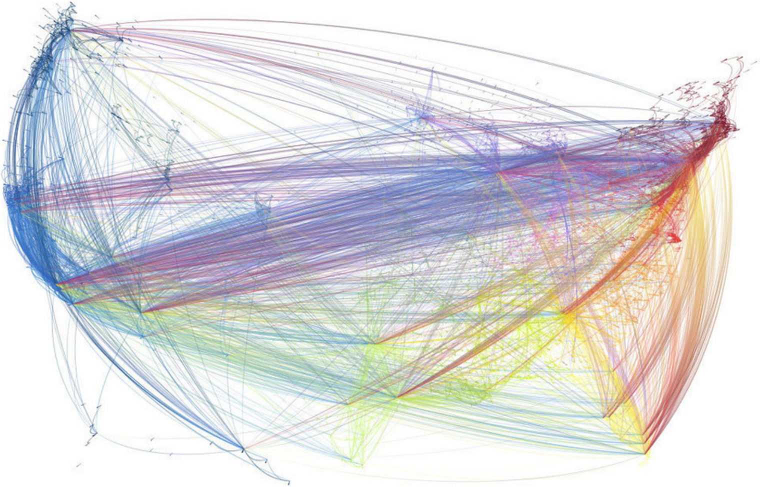
Wireshark screenshot of network traffic

Networking

- Hardware
- Protocols
- Software

The network is the computer. (John Gage)


C A N A D A
Pacific Ocean
Atlantic Ocean
Gulf Of Mexico
M E X I C O
US Department of Transportation
Federal Highway Administration
Office of Freight Management and Operations
Freight Analysis Framework
Estimated Average Annual Daily Truck Traffic
(1998)
Truck Volume Scale
50,000
25,000
12,500



Networking Options

Network type	maximum bandwidth (Mbits/second)	latency (microsecs)
Fast Ethernet	100	200
Gigabit Ethernet	1000	20–62
10 Gigabit Ethernet	10,000	4
Infiniband	2,000-290,000	0.5–5

Networking Options

Network type	maximum bandwidth (Mbits/second)	latency (microsecs)
Fast Ethernet	100	200
Gigabit Ethernet	1000	20–62
10 Gigabit Ethernet	10,000	4
Infiniband	2,000-290,000	0.5–5

Ethernet is the most cost-effective choice. Ethernet is a packet-based serial multi-drop network requiring no centralized control. All network access and arbitration control is performed by distributed mechanisms.

Networking Options

Network type	maximum bandwidth (Mbits/second)	latency (microsecs)
Fast Ethernet	100	200
Gigabit Ethernet	1000	20–62
10 Gigabit Ethernet	10,000	4
Infiniband	2,000-290,000	0.5–5

Ethernet is the most cost-effective choice. Ethernet is a packet-based serial multi-drop network requiring no centralized control. All network access and arbitration control is performed by distributed mechanisms.

Internet backbone uses specialized hardware and has speeds up to 500G/s. Experimental systems are available for much higher speeds.

Ethernet Protocol

- In an Ethernet network, machines use the Carrier Sense Multiple Access with Collision Detect (CSMA/CD) protocol for arbitration when multiple machines try to access the network at the same time.

Ethernet Protocol

- In an Ethernet network, machines use the Carrier Sense Multiple Access with Collision Detect (CSMA/CD) protocol for arbitration when multiple machines try to access the network at the same time.
- If packets collide, then the machines choose a random number from the interval $(0, k)$ and try again.

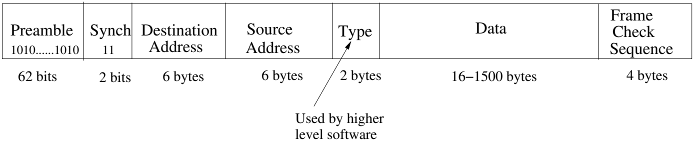
Ethernet Protocol

- In an Ethernet network, machines use the Carrier Sense Multiple Access with Collision Detect (CSMA/CD) protocol for arbitration when multiple machines try to access the network at the same time.
- If packets collide, then the machines choose a random number from the interval $(0, k)$ and try again.
- On subsequent collisions, the value k is doubled each time, making it a lot less likely that a collision would occur again. This is an example of an *exponential backoff protocol*.

Ethernet Packets

- The Ethernet packet has the format shown below.

Ethernet Packet Format



Ethernet Packets

- The Ethernet packet has the format shown below.

Ethernet Packet Format

Preamble 1010......1010	Synch 11	Destination Address	Source Address	Type	Data	Frame Check Sequence
62 bits	2 bits	6 bytes	6 bytes	2 bytes	16–1500 bytes	4 bytes

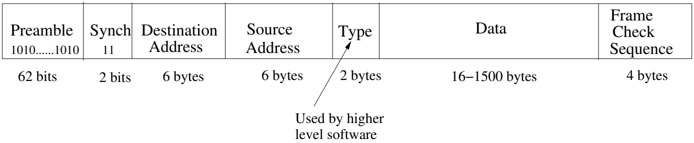
Used by higher
level software

- Note that the Maximum Transmission Unit (MTU) is 1500 bytes. Messages larger than that must be broken into smaller packets by higher layer network software.

Ethernet Packets

- The Ethernet packet has the format shown below.

Ethernet Packet Format



- Note that the Maximum Transmission Unit (MTU) is 1500 bytes. Messages larger than that must be broken into smaller packets by higher layer network software.
- Some switches accept Jumbo packets (up to 9000 bytes), which can improve the performance significantly.

Ethernet Packets

- The Ethernet packet has the format shown below.

Ethernet Packet Format

Preamble 1010......1010	Synch 11	Destination Address	Source Address	Type	Data	Frame Check Sequence
62 bits	2 bits	6 bytes	6 bytes	2 bytes	16–1500 bytes	4 bytes

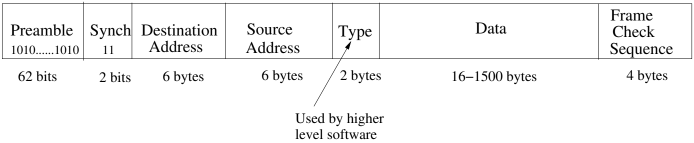
Used by higher level software

- Note that the Maximum Transmission Unit (MTU) is 1500 bytes. Messages larger than that must be broken into smaller packets by higher layer network software.
- Some switches accept Jumbo packets (up to 9000 bytes), which can improve the performance significantly.
- For many network drivers under Linux, the MTU can be set on the fly without reloading the driver!

Ethernet Packets

- The Ethernet packet has the format shown below.

Ethernet Packet Format



- Note that the Maximum Transmission Unit (MTU) is 1500 bytes. Messages larger than that must be broken into smaller packets by higher layer network software.
- Some switches accept Jumbo packets (up to 9000 bytes), which can improve the performance significantly.
- For many network drivers under Linux, the MTU can be set on the fly without reloading the driver!

Use the program `wireshark` to watch Ethernet packets on your network live!

Network Topology Design

Ranges of possibilities.

- Shared multi-drop passive cable, or

Network Topology Design

Ranges of possibilities.

- Shared multi-drop passive cable, or
- Tree structure of hubs and switches, or

Network Topology Design

Ranges of possibilities.

- Shared multi-drop passive cable, or
- Tree structure of hubs and switches, or
- Custom complicated switching technology, or

Network Topology Design

Ranges of possibilities.

- Shared multi-drop passive cable, or
- Tree structure of hubs and switches, or
- Custom complicated switching technology, or
- One big switch.

Network Topology Options

Hubs and Switches.

- Direct wire. Two machines can be connected directly by a Ethernet cable (usually a Cat 5e cable) without needing a hub or a switch. With multiple NICs per machine, we can create networks but then we need to specify routing tables to allow packets to get through. The machines will end up doing double-duty as routers.

Network Topology Options

Hubs and Switches.

- Direct wire. Two machines can be connected directly by a Ethernet cable (usually a Cat 5e cable) without needing a hub or a switch. With multiple NICs per machine, we can create networks but then we need to specify routing tables to allow packets to get through. The machines will end up doing double-duty as routers.
- Hubs and Repeaters All machines are visible from all machines and the CSMA/CD protocol is still used. A hub/repeater receives signals, cleans and amplifies, redistributes to all nodes.

Network Topology Options

Hubs and Switches.

- Direct wire. Two machines can be connected directly by a Ethernet cable (usually a Cat 5e cable) without needing a hub or a switch. With multiple NICs per machine, we can create networks but then we need to specify routing tables to allow packets to get through. The machines will end up doing double-duty as routers.
- Hubs and Repeaters All machines are visible from all machines and the CSMA/CD protocol is still used. A hub/repeater receives signals, cleans and amplifies, redistributes to all nodes.
- Switches. Accepts packets, interprets destination address fields and send packets down only the segment that has the destination node. Allows half the machines to communicate directly with the other half (subject to bandwidth constraints of the switch hardware). Multiple switches can be connected in a tree or sometimes other schemes. The root switch can become a bottleneck. The root switch can be a higher bandwidth switch.

Switches

Switches can be managed or unmanaged. Managed switches are more expensive but they also allow many useful configurations. Here are some examples.

Switches

Switches can be managed or unmanaged. Managed switches are more expensive but they also allow many useful configurations. Here are some examples.

- Port trunking (a.k.a Cisco EtherChannel). Allows up to 4 ports to be treated as one logical port. For example, this would allow a 4 Gbits/sec connection between two Gigabit switches.

Switches

Switches can be managed or unmanaged. Managed switches are more expensive but they also allow many useful configurations. Here are some examples.

- Port trunking (a.k.a Cisco EtherChannel). Allows up to 4 ports to be treated as one logical port. For example, this would allow a 4 Gbits/sec connection between two Gigabit switches.
- Linux Channel Bonding. Channel bonding means to bond together multiple NICs into one logical network connection. This requires the network switch to support some form of port trunking. Supported in the Linux kernel.

Switches

Switches can be managed or unmanaged. Managed switches are more expensive but they also allow many useful configurations. Here are some examples.

- Port trunking (a.k.a Cisco EtherChannel). Allows up to 4 ports to be treated as one logical port. For example, this would allow a 4 Gbits/sec connection between two Gigabit switches.
- Linux Channel Bonding. Channel bonding means to bond together multiple NICs into one logical network connection. This requires the network switch to support some form of port trunking. Supported in the Linux kernel.
- Switch Meshing. Allows up to 24 ports between switches to be treated as a single logical port, creating a very high bandwidth connection. useful for creating custom complicated topologies.

Switches

Switches can be managed or unmanaged. Managed switches are more expensive but they also allow many useful configurations. Here are some examples.

- Port trunking (a.k.a Cisco EtherChannel). Allows up to 4 ports to be treated as one logical port. For example, this would allow a 4 Gbits/sec connection between two Gigabit switches.
- Linux Channel Bonding. Channel bonding means to bond together multiple NICs into one logical network connection. This requires the network switch to support some form of port trunking. Supported in the Linux kernel.
- Switch Meshing. Allows up to 24 ports between switches to be treated as a single logical port, creating a very high bandwidth connection. useful for creating custom complicated topologies.
- Stackable, High bandwidth Switches. Stackable switches with special high bandwidth interconnect in-between the switches. For example, Cisco has 24-port Gigabit stackable switches with a 32 Gbits/sec interconnect. Up to 8 such switches can be stacked together. All the stacked switches can be controlled by one switch and managed as a single switch. If the controlling switch fails, the remaining switches hold an election and a new controlling switch is elected. Baystack also has stackable switches with a 40 Gbits/sec interconnect.

Network Interface Cards

- The Ethernet card, also known as the Network Interface Controller (NIC), contains the Data Link Layer and the Physical Layer (the two lowest layers of networking). Each Ethernet card has a unique hardware address that is know as its MAC address (MAC stands for Media Access Controller). The MAC address is usually printed on the Ethernet card.

Network Interface Cards

- The Ethernet card, also known as the Network Interface Controller (NIC), contains the Data Link Layer and the Physical Layer (the two lowest layers of networking). Each Ethernet card has a unique hardware address that is know as its MAC address (MAC stands for Media Access Controller). The MAC address is usually printed on the Ethernet card.
- The command `ifconfig` can be used to determine the MAC address from software. (Use `ipconfig` on Windows OS)

Network Interface Cards

- The Ethernet card, also known as the Network Interface Controller (NIC), contains the Data Link Layer and the Physical Layer (the two lowest layers of networking). Each Ethernet card has a unique hardware address that is know as its MAC address (MAC stands for Media Access Controller). The MAC address is usually printed on the Ethernet card.
- The command `ifconfig` can be used to determine the MAC address from software. (Use `ipconfig` on Windows OS)
- Another issue to consider is that having multi-processor boards may cause more load on the network cards in each node. Certain network cards have multiple network processors in them, making them better candidates for multi-processor motherboards.

Networking Models

- UUCP (Unix to Unix CoPy). Mostly over telephone lines to support mail and USENET news network. UUCP does not support remote login, rpc or distributed file systems.

Networking Models

- UUCP (Unix to Unix CoPy). Mostly over telephone lines to support mail and USENET news network. UUCP does not support remote login, rpc or distributed file systems.
- The ARPANET Reference Model (ARM) was the network model that led to the ISO OSI seven layer standardized model.

Networking Models

- UUCP (Unix to Unix CoPy). Mostly over telephone lines to support mail and USENET news network. UUCP does not support remote login, rpc or distributed file systems.
- The ARPANET Reference Model (ARM) was the network model that led to the ISO OSI seven layer standardized model.
- ISO Open System Interconnection (OSI). A reference model for networking prescribes seven layers of network protocols and strict methods of communication between them. Most systems implement simplified version of the OSI model. The ARPANET Reference Model (ARM) can be seen as a simplified OSI model.

Network Models (contd.)

ISO	ARM	4.2 BSD Layers	Example
application presentation session	process applications	user programs/libraries	ssh
		sockets	sock_stream
transport network data link	host–host network interface	protocols network interface	TCP/IP Ethernet driver
hardware	network hardware	network hardware	interlan controller

Protocols

- Protocols are the rules or standard that defines the syntax, semantics and synchronization of communication and possible error recovery methods. Protocols may be implemented by hardware, software, or a combination of both.

Protocols

- Protocols are the rules or standard that defines the syntax, semantics and synchronization of communication and possible error recovery methods. Protocols may be implemented by hardware, software, or a combination of both.
- Examples of protocols:
  - Internet Protocol (`IP`), Transmission Control Protocol (`TCP`), User Datagram Protocol (`UDP`)

Protocols

- Protocols are the rules or standard that defines the syntax, semantics and synchronization of communication and possible error recovery methods. Protocols may be implemented by hardware, software, or a combination of both.
- Examples of protocols:
  - Internet Protocol (`IP`), Transmission Control Protocol (`TCP`), User Datagram Protocol (`UDP`)
  - HyperText Transfer Protocol (`HTTP`), File Transfer Protocol (`FTP`)

Protocols

- Protocols are the rules or standard that defines the syntax, semantics and synchronization of communication and possible error recovery methods. Protocols may be implemented by hardware, software, or a combination of both.
- Examples of protocols:
  - Internet Protocol (`IP`), Transmission Control Protocol (`TCP`), User Datagram Protocol (`UDP`)
  - HyperText Transfer Protocol (`HTTP`), File Transfer Protocol (`FTP`)
  - Java Remote Method Invocation (`RMI`), Distributed Component Object Model (`DCOM`)

TCP/IP Addresses (1)

- The most prevalent protocol in networks is the `IP` (*Internet Protocol*). There are two higher-level protocols that run on top of the `IP` protocol. These are `TCP` (*Transmission Control Protocol*) and `UDP` (*User Datagram Protocol*).

TCP/IP Addresses (1)

- The most prevalent protocol in networks is the `IP` (*Internet Protocol*). There are two higher-level protocols that run on top of the `IP` protocol. These are `TCP` (*Transmission Control Protocol*) and `UDP` (*User Datagram Protocol*).
- Two versions of the IP protocol: `IPv4` protocol has 32-bit addresses while the `IPv6` protocol has 128-bit addresses. IP address range is divided into networks along an address bit boundary.

TCP/IP Addresses (1)

- The most prevalent protocol in networks is the `IP` (*Internet Protocol*). There are two higher-level protocols that run on top of the `IP` protocol. These are `TCP` (*Transmission Control Protocol*) and `UDP` (*User Datagram Protocol*).
- Two versions of the IP protocol: `IPv4` protocol has 32-bit addresses while the `IPv6` protocol has 128-bit addresses. IP address range is divided into networks along an address bit boundary.
- The portion of the address that remains fixed within a network is called the network address and the remainder is the host address.

TCP/IP Addresses (1)

- The most prevalent protocol in networks is the `IP` (*Internet Protocol*). There are two higher-level protocols that run on top of the `IP` protocol. These are `TCP` (*Transmission Control Protocol*) and `UDP` (*User Datagram Protocol*).
- Two versions of the IP protocol: `IPv4` protocol has 32-bit addresses while the `IPv6` protocol has 128-bit addresses. IP address range is divided into networks along an address bit boundary.
- The portion of the address that remains fixed within a network is called the network address and the remainder is the host address.
- The address with all 0's in the host address, for example 192.168.1.0, is the network address and cannot be assigned to any machine. The address with all 1's in the host address, for example 192.168.1.255, is the network broadcast address.

TCP/IP Addresses (2)

- A machine can refer to itself with the name localhost, which has a special IP address for it:
  `127.0.0.1` (IPv4)
  `::1` (IPv6)

TCP/IP Addresses (2)

- A machine can refer to itself with the name localhost, which has a special IP address for it:
  `127.0.0.1` (IPv4)
  `::1` (IPv6)
- Three IP ranges are reserved for private networks.
  - 10.0.0.0 – 10.255.255.255
  - 172.16.0.0 – 172.31.255.255
  - 192.168.0.0 – 192.168.255.255

TCP/IP Addresses (2)

- A machine can refer to itself with the name localhost, which has a special IP address for it:
  `127.0.0.1` (IPv4)
  `::1` (IPv6)
- Three IP ranges are reserved for private networks.
  - 10.0.0.0 – 10.255.255.255
  - 172.16.0.0 – 172.31.255.255
  - 192.168.0.0 – 192.168.255.255
- These addresses are permanently unassigned, not forwarded by Internet backbone routers and thus do not conflict with publicly addressable IP addresses.

Sockets

- A socket is an endpoint of communication.

Sockets

- A socket is an endpoint of communication.
- A socket in use usually has an address bound to it.

Sockets

- A socket is an endpoint of communication.
- A socket in use usually has an address bound to it.
- The nature of the address depends upon the communication domain of the socket.

Sockets

- A socket is an endpoint of communication.
- A socket in use usually has an address bound to it.
- The nature of the address depends upon the communication domain of the socket.
- Processes communicating in the same domain use the same address format.

Sockets

- A socket is an endpoint of communication.
- A socket in use usually has an address bound to it.
- The nature of the address depends upon the communication domain of the socket.
- Processes communicating in the same domain use the same address format.

Typical communication domains:

domain type	symbolic name	address format
Unix domain	`AF_Unix`	pathnames
Internet domain	`AF_INET`	Internet address and port number

Types of Sockets

- *Stream sockets.* Reliable, duplex, sequenced data streams. e.g. pipes, TCP protocol

Types of Sockets

- *Stream sockets.* Reliable, duplex, sequenced data streams. e.g. pipes, TCP protocol
- *Sequenced packet sockets.* Reliable, duplex, record boundaries

Types of Sockets

- *Stream sockets.* Reliable, duplex, sequenced data streams. e.g. pipes, TCP protocol
- *Sequenced packet sockets.* Reliable, duplex, record boundaries
- *Datagram sockets.* Unreliable, unsequenced, variable size packets

Types of Sockets

- *Stream sockets.* Reliable, duplex, sequenced data streams. e.g. pipes, TCP protocol
- *Sequenced packet sockets.* Reliable, duplex, record boundaries
- *Datagram sockets.* Unreliable, unsequenced, variable size packets
- *Reliably delivered message sockets.*

Types of Sockets

- *Stream sockets.* Reliable, duplex, sequenced data streams. e.g. pipes, TCP protocol
- *Sequenced packet sockets.* Reliable, duplex, record boundaries
- *Datagram sockets.* Unreliable, unsequenced, variable size packets
- *Reliably delivered message sockets.*
- *Raw sockets.* Allows access to TCP, IP or Ethernet protocol

Socket System Calls (for C)

- A `socket()` system call creates a socket. It returns a *socket descriptor*, similar to a file descriptor. The socket descriptor indexes the open file table.

Socket System Calls (for C)

- A `socket()` system call creates a socket. It returns a *socket descriptor*, similar to a file descriptor. The socket descriptor indexes the open file table.
- For another process to access a socket, it must have a name. A name is bound to a socket by the `bind()` system call.

Socket System Calls (for C)

- A `socket()` system call creates a socket. It returns a *socket descriptor*, similar to a file descriptor. The socket descriptor indexes the open file table.
- For another process to access a socket, it must have a name. A name is bound to a socket by the `bind()` system call.
- `connect(...)` Connects a process to a socket with a well-known address (e.g. in the AF_INET domain, the address is the IP address and a port number).

Socket System Calls (for C)

- A `socket()` system call creates a socket. It returns a *socket descriptor*, similar to a file descriptor. The socket descriptor indexes the open file table.
- For another process to access a socket, it must have a name. A name is bound to a socket by the `bind()` system call.
- `connect(...)` Connects a process to a socket with a well-known address (e.g. in the AF_INET domain, the address is the IP address and a port number).
- `listen(...)` Informs the operating system that the process is willing to accept connections to a socket that has been bound. Has an argument that lets the operating system know how many connections to queue up.

Socket System Calls (for C)

- A `socket()` system call creates a socket. It returns a *socket descriptor*, similar to a file descriptor. The socket descriptor indexes the open file table.
- For another process to access a socket, it must have a name. A name is bound to a socket by the `bind()` system call.
- `connect(...)` Connects a process to a socket with a well-known address (e.g. in the AF_INET domain, the address is the IP address and a port number).
- `listen(...)` Informs the operating system that the process is willing to accept connections to a socket that has been bound. Has an argument that lets the operating system know how many connections to queue up.
- `accept(...)` Normally a blocking call. When a connection is made, returns with a new socket id.

Socket System Calls (for C)

- A `socket()` system call creates a socket. It returns a *socket descriptor*, similar to a file descriptor. The socket descriptor indexes the open file table.
- For another process to access a socket, it must have a name. A name is bound to a socket by the `bind()` system call.
- `connect(...)` Connects a process to a socket with a well-known address (e.g. in the AF_INET domain, the address is the IP address and a port number).
- `listen(...)` Informs the operating system that the process is willing to accept connections to a socket that has been bound. Has an argument that lets the operating system know how many connections to queue up.
- `accept(...)` Normally a blocking call. When a connection is made, returns with a new socket id.
- `close(...)`, `shutdown(...)`: Close both or one end of a socket respectively.

Client-Server Setup Using Sockets

Server side	Client side
socket(...)	
bind(...)	socket(...)
listen(...)	connect(...)
accept(...)	
read/write	read/write
close/shutdown	

TCP/IP and Linux/Unix Networking

- Port numbers in the range 1-255 are reserved in TCP/IP protocols for well known servers. In addition, Linux/Unix reserve the ports 1-1023 for superuser processes. Ports from 1024 to 65535 are available for user processes.

TCP/IP and Linux/Unix Networking

- Port numbers in the range 1-255 are reserved in TCP/IP protocols for well known servers. In addition, Linux/Unix reserve the ports 1-1023 for superuser processes. Ports from 1024 to 65535 are available for user processes.
- The file `/etc/services` contains the port numbers for well known servers. For example:
  - port 37 is reserved for getting the time from a system
  - port 7 is used for echoing the data sent by a client back
  - port 21 is used by the FTP (File Transfer Protocol) client/server
  - port 22 is used by SSH (Secure Shell Protocol) client/server
  - port 80 is used by the HTTP (HyperText Transfer Protocol) daemon
  - port 443 is used by the HTTP over TLS/SSL (Transport Layer Security/Secure Sockets Layer for secure connections) (represented as the `https://` protocol)

TCP/IP and Linux/Unix Networking

- Port numbers in the range 1-255 are reserved in TCP/IP protocols for well known servers. In addition, Linux/Unix reserve the ports 1-1023 for superuser processes. Ports from 1024 to 65535 are available for user processes.
- The file `/etc/services` contains the port numbers for well known servers. For example:
  - port 37 is reserved for getting the time from a system
  - port 7 is used for echoing the data sent by a client back
  - port 21 is used by the FTP (File Transfer Protocol) client/server
  - port 22 is used by SSH (Secure Shell Protocol) client/server
  - port 80 is used by the HTTP (HyperText Transfer Protocol) daemon
  - port 443 is used by the HTTP over TLS/SSL (Transport Layer Security/Secure Sockets Layer for secure connections) (represented as the `https://` protocol)
- The configuration directory `/etc/xinetd.d/` contains several files, one per service type, that control what is provided by the internet super-daemon `xinetd` under Linux. The super-daemon listens on all ports and invoked secondary daemons as needed.

TCP/IP and Linux/Unix Networking

- Port numbers in the range 1-255 are reserved in TCP/IP protocols for well known servers. In addition, Linux/Unix reserve the ports 1-1023 for superuser processes. Ports from 1024 to 65535 are available for user processes.
- The file `/etc/services` contains the port numbers for well known servers. For example:
  - port 37 is reserved for getting the time from a system
  - port 7 is used for echoing the data sent by a client back
  - port 21 is used by the FTP (File Transfer Protocol) client/server
  - port 22 is used by SSH (Secure Shell Protocol) client/server
  - port 80 is used by the HTTP (HyperText Transfer Protocol) daemon
  - port 443 is used by the HTTP over TLS/SSL (Transport Layer Security/Secure Sockets Layer for secure connections) (represented as the `https://` protocol)
- The configuration directory `/etc/xinetd.d/` contains several files, one per service type, that control what is provided by the internet super-daemon `xinetd` under Linux. The super-daemon listens on all ports and invoked secondary daemons as needed. *What is the advantage of such a setup?*

Servers

Design types:

- A single-threaded server handles one connection at a time.

Servers

Design types:

- A single-threaded server handles one connection at a time.
- A multi-threaded server accepts connections and passes them off to their own threads for processing.

Servers

Design types:

- A single-threaded server handles one connection at a time.
- A multi-threaded server accepts connections and passes them off to their own threads for processing.
- A multi-process server forks off a copy of itself after a connection to handle the client, while the original server process goes back to accept further connections.

Client/Server Examples in C

The examples are in folder `sockets-C/tcp`

- Single-threaded server: timeserver.c and client timeclient.c

Client/Server Examples in C

The examples are in folder `sockets-C/tcp`

- Single-threaded server: timeserver.c and client timeclient.c
- Multi-process server: tcpserver.c and client tcpclient.c

Client/Server Examples in C

The examples are in folder `sockets-C/tcp`

- Single-threaded server: timeserver.c and client timeclient.c
- Multi-process server: tcpserver.c and client tcpclient.c
- Multi-threaded server: Left as an exercise....

Client/Server Examples in C

The examples are in folder `sockets-C/tcp`

- Single-threaded server: timeserver.c and client timeclient.c
- Multi-process server: tcpserver.c and client tcpclient.c
- Multi-threaded server: Left as an exercise....
- Note that read/write on sockets is slightly different than read/write on files. A read/write on a socket may return a count less than asked for. This is not an error since with sockets, the buffer in the kernel may be full. We can just keep calling read/write until the right amount of data has been read or written.

Useful Tools

- Use `netstat -ni` to find information on the network interfaces.
- Use `netstat -rn` to see the routing table.
- Use `netstat -nap` to see the processes that are using specific interfaces and ports. You need to be superuser to be able to see complete process information. Nice way of determining who has a port bound up!
- Use `netstat -s` to see a summary of network statistics. For example, `netstat -s -udp` summarizes all UDP traffic.

Useful Tools

- Use `netstat -ni` to find information on the network interfaces.
- Use `netstat -rn` to see the routing table.
- Use `netstat -nap` to see the processes that are using specific interfaces and ports. You need to be superuser to be able to see complete process information. Nice way of determining who has a port bound up!
- Use `netstat -s` to see a summary of network statistics. For example, `netstat -s -udp` summarizes all UDP traffic.
- Use `lsof -w -n -i tcp:<port num>` gives the pid of the process using that port so you can kill a process that didn't clean up or properly release the port.

Useful Tools

- Use `netstat -ni` to find information on the network interfaces.
- Use `netstat -rn` to see the routing table.
- Use `netstat -nap` to see the processes that are using specific interfaces and ports. You need to be superuser to be able to see complete process information. Nice way of determining who has a port bound up!
- Use `netstat -s` to see a summary of network statistics. For example, `netstat -s -udp` summarizes all UDP traffic.
- Use `lsof -w -n -i tcp:<port num>` gives the pid of the process using that port so you can kill a process that didn't clean up or properly release the port.
- Use `/sbin/ifconfig eth0` to get details on the interface `eth0`. Running `ifconfig` without any options gives details on all interfaces.

Useful Tools

- Use `netstat -ni` to find information on the network interfaces.
- Use `netstat -rn` to see the routing table.
- Use `netstat -nap` to see the processes that are using specific interfaces and ports. You need to be superuser to be able to see complete process information. Nice way of determining who has a port bound up!
- Use `netstat -s` to see a summary of network statistics. For example, `netstat -s -udp` summarizes all UDP traffic.
- Use `lsof -w -n -i tcp:<port num>` gives the pid of the process using that port so you can kill a process that didn't clean up or properly release the port.
- Use `/sbin/ifconfig eth0` to get details on the interface `eth0`. Running `ifconfig` without any options gives details on all interfaces.
- Use `ping` to check if a machine is alive. Use ping -b with a network address to find all machines on a local area network.

Useful Tools

- Use `netstat -ni` to find information on the network interfaces.
- Use `netstat -rn` to see the routing table.
- Use `netstat -nap` to see the processes that are using specific interfaces and ports. You need to be superuser to be able to see complete process information. Nice way of determining who has a port bound up!
- Use `netstat -s` to see a summary of network statistics. For example, `netstat -s` -udp summarizes all UDP traffic.
- Use `lsof -w -n -i tcp:<port num>` gives the pid of the process using that port so you can kill a process that didn't clean up or properly release the port.
- Use `/sbin/ifconfig eth0` to get details on the interface `eth0`. Running `ifconfig` without any options gives details on all interfaces.
- Use `ping` to check if a machine is alive. Use ping -b with a network address to find all machines on a local area network.
- Use `wireshark` to watch network packets in real time! You will need superuser access to be able to use `wireshark` fully.

Client/Server Communication Using Sockets in Java

- A server creates a ServerSocket object for a specific port and uses the `accept()` method to wait for a connection.

Client/Server Communication Using Sockets in Java

- A server creates a ServerSocket object for a specific port and uses the `accept()` method to wait for a connection.
- The client uses (hostname, port number) pair to locate the server.

Client/Server Communication Using Sockets in Java

- A server creates a ServerSocket object for a specific port and uses the `accept()` method to wait for a connection.
- The client uses (hostname, port number) pair to locate the server.
- The server accepts the client's request and creates a `Socket` object for communicating with the client. There is a separate `Socket` object created for each client request accepted.

Client/Server Communication Using Sockets in Java

- A server creates a ServerSocket object for a specific port and uses the `accept()` method to wait for a connection.
- The client uses (hostname, port number) pair to locate the server.
- The server accepts the client's request and creates a `Socket` object for communicating with the client. There is a separate `Socket` object created for each client request accepted.
- Now the server and the client can read/write to the streams associated with the sockets.

Client/Server Communication Using Sockets in Java

- A server creates a ServerSocket object for a specific port and uses the `accept()` method to wait for a connection.
- The client uses (hostname, port number) pair to locate the server.
- The server accepts the client's request and creates a `Socket` object for communicating with the client. There is a separate `Socket` object created for each client request accepted.
- Now the server and the client can read/write to the streams associated with the sockets.
- Always open `OutputStream` before `InputStream` on a socket to avoid deadlock and synchronization problems.

Client Example (Java)

```java
try {
    Socket server = new Socket("www.party.com",1234);
    OutputStream out = server.getOutputStream();
    InputStream in = server.getInputStream();

    out.write(42); //write a byte

    //write a newline or carriage return delimited string
    PrintWriter pout = new PrintWriter(out, true);
    pout.println("Hello!");

    //read a byte
    Byte response = in.read();

    //read a newline or carriage return delimited string
    BufferedReader bin = new BufferedReader (new
   InputStreamReader(in));
    String answer = bin.readLine();

    //send a serialized Java object
    ObjectOutputStream oout = new ObjectOutputStream(out);
    oout.writeObject(new java.util.Date());
    oout.flush();

    server.close();
} catch (IOException e) {System.err.println(e);}
```

Server Example (Java)

```
try {//meanwhile, on www.party.com...
    ServerSocket listener = new ServerSocket(1234);
    while (!finished) {
        Socket client = listener.accept();
        InputStream in = client.getInputStream();
        OutputStream out = client.getOutputStream();

        Byte someByte = in.read(); //read a byte

        //read a newline/carriage return delimited string
        BufferedReader bin =
             new BufferedReader(new InputStreamReader(in));
        String someString = bin.readLine();

        out.write(42); //write a byte
        PrintWriter pout = new PrintWriter(out, true);
        pout.println("Goodbye!");

        //read a serialized Java object
        ObjectInputStream oin = new ObjectInputStream(in);
        Date date = (Date) oin.readObject();

        client.close();
        //...
    }
    listener.close();
} catch (IOException e) {System.err.println(e);}
```

TCP examples in Java

See the folder and subfolders in `sockets`.

- A remote date client DateAtHost.java in package `tcp.client`.

TCP examples in Java

See the folder and subfolders in `sockets`.

- A remote date client DateAtHost.java in package `tcp.client`.
- Single-threaded server and client: See TimeServer.java and TimeClient.java in package `tcp.singlethreaded`.

TCP examples in Java

See the folder and subfolders in `sockets`.

- A remote date client DateAtHost.java in package `tcp.client`.
- Single-threaded server and client: See TimeServer.java and TimeClient.java in package `tcp.singlethreaded`.
- Multi-threaded server and client: See TimeServer.java and TimeClient.java in package `tcp.multithreaded`.

TCP examples in Java

See the folder and subfolders in `sockets`.

- A remote date client DateAtHost.java in package `tcp.client`.
- Single-threaded server and client: See TimeServer.java and TimeClient.java in package `tcp.singlethreaded`.
- Multi-threaded server and client: See TimeServer.java and TimeClient.java in package `tcp.multithreaded`.
- The `SecurityManager` can impose arbitrary restrictions on applications as to what hosts they may or may not talk to, and whether they can listen for connections.

TCP examples in Java

See the folder and subfolders in `sockets`.

- A remote date client DateAtHost.java in package `tcp.client`.
- Single-threaded server and client: See TimeServer.java and TimeClient.java in package `tcp.singlethreaded`.
- Multi-threaded server and client: See TimeServer.java and TimeClient.java in package `tcp.multithreaded`.
- The `SecurityManager` can impose arbitrary restrictions on applications as to what hosts they may or may not talk to, and whether they can listen for connections.
- A server could run a proxy that lets the application communicate indirectly with anyone the server likes and allows.

TCP examples in Java

See the folder and subfolders in `sockets`.

- A remote date client DateAtHost.java in package `tcp.client`.
- Single-threaded server and client: See TimeServer.java and TimeClient.java in package `tcp.singlethreaded`.
- Multi-threaded server and client: See TimeServer.java and TimeClient.java in package `tcp.multithreaded`.
- The `SecurityManager` can impose arbitrary restrictions on applications as to what hosts they may or may not talk to, and whether they can listen for connections.
- A server could run a proxy that lets the application communicate indirectly with anyone the server likes and allows. *How would you design a proxy server?*

Object Based Server/Clients Example

- The client will send a serialized object to the server. This object represents a request. The server will send an serialized object back as an reply that represents the response.

Object Based Server/Clients Example

- The client will send a serialized object to the server. This object represents a request. The server will send an serialized object back as an reply that represents the response.
- We will use a base class `Request` for the various kinds of requests.

```
public class Request implements java.io.Serializable {}

public class DateRequest extends Request {}

public class WorkRequest extends Request {
    public Object execute() {return null;}
}
```

Object Based Server/Clients Example

- The client will send a serialized object to the server. This object represents a request. The server will send an serialized object back as an reply that represents the response.
- We will use a base class Request for the various kinds of requests.

```
public class Request implements java.io.Serializable {}

public class DateRequest extends Request {}

public class WorkRequest extends Request {
    public Object execute() {return null;}
}
```

- The client sends a WorkRequest object to the server to get the server to perform work for the client. The server calls the request object's execute method and returns the resulting object as a response.
- A sample work request class: MyCalculation.java

Object Based Server/Clients Example

- The client will send a serialized object to the server. This object represents a request. The server will send an serialized object back as an reply that represents the response.
- We will use a base class Request for the various kinds of requests.

```
public class Request implements java.io.Serializable {}

public class DateRequest extends Request {}

public class WorkRequest extends Request {
    public Object execute() {return null;}
}
```

- The client sends a WorkRequest object to the server to get the server to perform work for the client. The server calls the request object's execute method and returns the resulting object as a response.
- A sample work request class: MyCalculation.java
- The Server.java, Client.java that ties it all together.

Running the Object Server/Client

- Start the server on one host:
  `on plainoldearth.net:  java Server 1234`

Running the Object Server/Client

- Start the server on one host:
  `on plainoldearth.net:  java Server 1234`
- Start the client anywhere on the Internet. E.g. on `restaurant.endofuniverse.net`:
  `java Client plainoldearth.net 1234`

Running the Object Server/Client

- Start the server on one host:
  `on plainoldearth.net:  java Server 1234`
- Start the client anywhere on the Internet. E.g. on `restaurant.endofuniverse.net`:
  `java Client plainoldearth.net 1234`
- Note that the server machine must have all the classes that the client has in order to be able to execute them on the client's behalf. That may be an unreasonable assumption since you may want to serve many kinds of clients without having to store all their classes.

Running the Object Server/Client

- Start the server on one host:
  `on plainoldearth.net:  java Server 1234`
- Start the client anywhere on the Internet. E.g. on `restaurant.endofuniverse.net`:
  `java Client plainoldearth.net 1234`
- Note that the server machine must have all the classes that the client has in order to be able to execute them on the client's behalf. That may be an unreasonable assumption since you may want to serve many kinds of clients without having to store all their classes.
- It is possible to overcome this restriction with the help of reflection. (More on this later)

In-class Exercise

- Sketch the setup for two servers that act as both server and client with each other.

In-class Exercise

- Sketch the setup for two servers that act as both server and client with each other.
- How would you implement two processes playing ping pong across the network? Are they both servers? Are they both clients?

Socket and ServerSocket Options

- `ServerSocket` and `Socket` classes have several useful options.
- For example: we can set a timeout on a socket, we can set the receive buffer sizes, etc.
- See examples ServerSocketOptions.java and ClientSocketOptions.java in the package `tcp.socketoptions`.

Datagram Sockets

- A datagram is a discrete amount of data transmitted in one chunk or packet.

Datagram Sockets

- A datagram is a discrete amount of data transmitted in one chunk or packet.
- Datagrams are not guaranteed to be delivered, nor are they guaranteed to arrive in the right order. Even duplicate datagrams might arrive.

Datagram Sockets

- A datagram is a discrete amount of data transmitted in one chunk or packet.
- Datagrams are not guaranteed to be delivered, nor are they guaranteed to arrive in the right order. Even duplicate datagrams might arrive.
- Datagrams use the `UDP` protocol, which is more lightweight than the `TCP` protocol.

Datagram Sockets

- A datagram is a discrete amount of data transmitted in one chunk or packet.
- Datagrams are not guaranteed to be delivered, nor are they guaranteed to arrive in the right order. Even duplicate datagrams might arrive.
- Datagrams use the `UDP` protocol, which is more lightweight than the `TCP` protocol.
- Domain Name Service (`DNS`) and Network File System (`NFS`) use UDP.

Datagram Sockets

- A datagram is a discrete amount of data transmitted in one chunk or packet.
- Datagrams are not guaranteed to be delivered, nor are they guaranteed to arrive in the right order. Even duplicate datagrams might arrive.
- Datagrams use the `UDP` protocol, which is more lightweight than the `TCP` protocol.
- Domain Name Service (`DNS`) and Network File System (`NFS`) use UDP.
- Example: UdpServer1.java and UdpClient1.java in package `tcp.udp`.

Example Network Applications

- How to implement a chat program?

Example Network Applications

- How to implement a chat program?
- How to implement a chat program with photos, video and sound?

Example Network Applications

- How to implement a chat program?
- How to implement a chat program with photos, video and sound?
- How to implement a web server?

The HTTP Protocol and Web Servers

- A Web Server implements at least the `HTTP` protocol. In order to talk to a Web server, a client program (e.g. a web browser) must speak the HTTP protocol.

The HTTP Protocol and Web Servers

- A Web Server implements at least the `HTTP` protocol. In order to talk to a Web server, a client program (e.g. a web browser) must speak the HTTP protocol.
- Details of the `HTTP` protocol can be found at the home page for the World Wide Web consortium (`www.w3.org`).

The HTTP Protocol and Web Servers

- A Web Server implements at least the HTTP protocol. In order to talk to a Web server, a client program (e.g. a web browser) must speak the HTTP protocol.
- Details of the HTTP protocol can be found at the home page for the World Wide Web consortium (www.w3.org).
- Requests/Methods in the HTTP Protocol:

```
GET <pathname> HTTP/x.y  (e.g. GET /sample.html HTTP/1.0)
HEAD <pathname> (same as GET except only metadata is returned)
POST <string>
       (the server accepts the entity enclosed in the request)
       (useful for running server side scripts)
```

The HTTP Protocol and Web Servers

- A Web Server implements at least the HTTP protocol. In order to talk to a Web server, a client program (e.g. a web browser) must speak the HTTP protocol.
- Details of the HTTP protocol can be found at the home page for the World Wide Web consortium (www.w3.org).
- Requests/Methods in the HTTP Protocol:

```
GET <pathname> HTTP/x.y  (e.g. GET /sample.html HTTP/1.0)
HEAD <pathname> (same as GET except only metadata is returned)
POST <string>
       (the server accepts the entity enclosed in the request)
       (useful for running server side scripts)
```

- Response from server:

```
HTTP-Version status-code reason-phrase <CR><LF>
```

Selected status codes in the HTTP Protocol

Status codes:

```
200 OK
201 Created

301 Moved permanently
305 Use Proxy
307 Temporary redirect

400 Bad request (bad syntax)
401 Unauthorized
402 Payment required
403 Forbidden
404 Not found

500 Internal server error
501 Not implemented
503 Service unavailable
505 HTTP version not supported
```

A Tiny Web Server

- Example: TinyHttpd.java, Client.java

A Tiny Web Server

- Example: TinyHttpd.java, Client.java
- To run the server, go to the folder `sockets` folder and run the following command:
  `java tinyhttpd.TinyHttpd 5005`

A Tiny Web Server

- Example: TinyHttpd.java, Client.java
- To run the server, go to the folder `sockets` folder and run the following command:
  `java tinyhttpd.TinyHttpd 5005`
- Then use a web browser and point to `localhost:5005` or use the provided client program.
  `java tinyhttpd.Client localhost 5005`

A Tiny Web Server

- Example: TinyHttpd.java, Client.java
- To run the server, go to the folder `sockets` folder and run the following command:
  `java tinyhttpd.TinyHttpd 5005`
- Then use a web browser and point to `localhost:5005` or use the provided client program.
  `java tinyhttpd.Client localhost 5005`
- Warning! This web server will serve files without any protection from a system!

Using the Built-In Security Manager

- Java has a built-in security manager, which if activated gives the same level of access as given to applets (that is, not much). The security manager can be activated with a command line option.
  `java -Djava.security.manager TinyHttpd`

Using the Built-In Security Manager

- Java has a built-in security manager, which if activated gives the same level of access as given to applets (that is, not much). The security manager can be activated with a command line option.
  `java -Djava.security.manager TinyHttpd`
- However, we want to give access to create and use sockets. So we create a policy file (using the tool `policytool` that comes with the Java toolkit).

```
grant {
        permission java.net.SocketPermission
                "*:1024-", "listen,accept,connect";
};
```

Using the Built-In Security Manager

- Java has a built-in security manager, which if activated gives the same level of access as given to applets (that is, not much). The security manager can be activated with a command line option.
  `java -Djava.security.manager TinyHttpd`
- However, we want to give access to create and use sockets. So we create a policy file (using the tool `policytool` that comes with the Java toolkit).

```
grant {
        permission java.net.SocketPermission
                "*:1024-", "listen,accept,connect";
};
```

- Add the following after the catch for `FileNotFoundException`.

```
catch (SecurityException e) {pout.println("403 Forbidden");}
```

Using the Built-In Security Manager

- Java has a built-in security manager, which if activated gives the same level of access as given to applets (that is, not much). The security manager can be activated with a command line option.
  `java -Djava.security.manager TinyHttpd`
- However, we want to give access to create and use sockets. So we create a policy file (using the tool `policytool` that comes with the Java toolkit).

```
grant {
        permission java.net.SocketPermission
                "*:1024-", "listen,accept,connect";
};
```

- Add the following after the catch for `FileNotFoundException`.

```
catch (SecurityException e) {pout.println("403 Forbidden");}
```

- Recompile and run the server as follows.
  `java -Djava.security.manager`
  `-Djava.security.policy=mysecurity.policy TinyHttpd 5005`

Adding a Custom Security Manager to TinyHttpd

```
class TinyHttpdSecurityManager extends SecurityManager {
  public void checkAccess(Thread g) {};
  public void checkListen(int port) {};
  public void checkLink(String lib) {};
  public void checkPropertyAccess(String key) {};
  public void checkAccept(String host, int port) {};
  public void checkWrite(FileDescriptor fd) {};
  public void checkRead(FileDescriptor fd ) {};

  public void checkRead(String s) {
    if (new File(s).isAbsolute() || (s.indexOf("..") != -1))
      throw new SecurityException("Access to file: " + s
                                              + " denied.");
  }
}
//add the following to the TinyHttpd at the start of the
//main method but after creating the ServerSocket

System.setSecurityManager(new TinyHttpdSecurityManager());
```

Suggestions for Improvement

- Use a buffer and send large amount of data in several passes.

Suggestions for Improvement

- Use a buffer and send large amount of data in several passes.
- Generate linked listings for directories (if no index.html was found).

Suggestions for Improvement

- Use a buffer and send large amount of data in several passes.
- Generate linked listings for directories (if no index.html was found).
- Log all requests in a log file. A sample entry is shown below (taken from the access log of Apache web server):

```
66.249.69.87 - - [28/Jan/2016:23:16:04 -0700]
 "GET /~amit/teaching/555/cs455-555.html HTTP/1.1" 200 7343 "-"
 "Mozilla/5.0 (compatible; Googlebot/2.1;
  +http://www.google.com/bot.html)"
```

Suggestions for Improvement

- Use a buffer and send large amount of data in several passes.
- Generate linked listings for directories (if no index.html was found).
- Log all requests in a log file. A sample entry is shown below (taken from the access log of Apache web server):

```
66.249.69.87 - - [28/Jan/2016:23:16:04 -0700]
 "GET /~amit/teaching/555/cs455-555.html HTTP/1.1" 200 7343 "-"
 "Mozilla/5.0 (compatible; Googlebot/2.1;
  +http://www.google.com/bot.html)"
```

- Add other kinds of requests (other than GET)

Suggestions for Improvement

- Use a buffer and send large amount of data in several passes.
- Generate linked listings for directories (if no index.html was found).
- Log all requests in a log file. A sample entry is shown below (taken from the access log of Apache web server):

```
66.249.69.87 - - [28/Jan/2016:23:16:04 -0700]
 "GET /~amit/teaching/555/cs455-555.html HTTP/1.1" 200 7343 "-"
 "Mozilla/5.0 (compatible; Googlebot/2.1;
  +http://www.google.com/bot.html)"
```

- Add other kinds of requests (other than GET)
- Use the Executor interface to manage the threads

Suggestions for Improvement

- Use a buffer and send large amount of data in several passes.
- Generate linked listings for directories (if no index.html was found).
- Log all requests in a log file. A sample entry is shown below (taken from the access log of Apache web server):

```
66.249.69.87 - - [28/Jan/2016:23:16:04 -0700]
 "GET /~amit/teaching/555/cs455-555.html HTTP/1.1" 200 7343 "-"
 "Mozilla/5.0 (compatible; Googlebot/2.1;
  +http://www.google.com/bot.html)"
```

- Add other kinds of requests (other than GET)
- Use the Executor interface to manage the threads
- Use scalable I/O with `java.nio` package.

Scalable I/O with java.nio package

- Nonblocking and selectable network communications are used to create services that can handle high volumes of simultaneous client requests.

Scalable I/O with java.nio package

- Nonblocking and selectable network communications are used to create services that can handle high volumes of simultaneous client requests.
- Starting one thread per client request can consume a lot of resources. One strategy is to use nonblocking I/O operations to manage a lot of communications from a single thread. The second strategy is to use a configurable pool of threads, taking advantage of machines with many processors.

Scalable I/O with java.nio package

- Nonblocking and selectable network communications are used to create services that can handle high volumes of simultaneous client requests.
- Starting one thread per client request can consume a lot of resources. One strategy is to use nonblocking I/O operations to manage a lot of communications from a single thread. The second strategy is to use a configurable pool of threads, taking advantage of machines with many processors.
- The `java.nio` package provides selectable channels. A selectable channel allows for the registration of a special kind of listener called a selector that can check the readiness of the channel for operations such as reading and writing or accepting or creating network connections.

Selectable Channels

- Create a selector object.

```
Selector selector = Selector.open();
```

Selectable Channels

- Create a selector object.

```
Selector selector = Selector.open();
```

- To register one or more channels with the selector, set them to nonblocking mode.

```
SelectableChannel channelA = ...;
channelA.configureBlocking(false);
```

Selectable Channels

- Create a selector object.

```
Selector selector = Selector.open();
```

- To register one or more channels with the selector, set them to nonblocking mode.

```
SelectableChannel channelA = ...;
channelA.configureBlocking(false);
```

- Then, we register the channels.

```
int interestOps = SelectionKey.OP_READ | SelectionKey.OP_WRITE;
SelectionKey key = channelA.register(selector, interestOps);
```

Selectable Channels

- Create a selector object.

```
Selector selector = Selector.open();
```

- To register one or more channels with the selector, set them to nonblocking mode.

```
SelectableChannel channelA = ...;
channelA.configureBlocking(false);
```

- Then, we register the channels.

```
int interestOps = SelectionKey.OP_READ | SelectionKey.OP_WRITE;
SelectionKey key = channelA.register(selector, interestOps);
```

- The possible values of interest ops are: `OP_READ`, `OP_WRITE`, `OP_CONNECT` and `OP_ACCEPT`. These values can be OR'd together to express interest in one or more operations.

Selectable Channels

- Create a selector object.

```
Selector selector = Selector.open();
```

- To register one or more channels with the selector, set them to nonblocking mode.

```
SelectableChannel channelA = ...;
channelA.configureBlocking(false);
```

- Then, we register the channels.

```
int interestOps = SelectionKey.OP_READ | SelectionKey.OP_WRITE;
SelectionKey key = channelA.register(selector, interestOps);
```

- The possible values of interest ops are: `OP_READ`, `OP_WRITE`, `OP_CONNECT` and `OP_ACCEPT`. These values can be OR'd together to express interest in one or more operations.
- Once one or more channels are registered with the Selector, we can perform a select operations by using one of the `select()` methods.

```
int readyCount = selector.select(); //block until a channel is ready
int readyCount = selector.selectNow(); //returns immediately
int readyCount = selector.select(50); //timeout of 50 millisecs

while (selector.select(50) == 0);
```

Checking for ready channels

Once `select()` comes back with a non-zero ready count, then we can get the set of ready channels from the `Selector` with the `selectedKeys()` method and iterate through them.

Checking for ready channels

Once `select()` comes back with a non-zero ready count, then we can get the set of ready channels from the `Selector` with the `selectedKeys()` method and iterate through them.

```
Set readySet = selector.selectedKeys();
for (Iterator itr = readySet.iterator(); itr.hasNext();) {
    SelectionKey key = (SelectionKey) itr.next();
    itr.remove(); //remove the key from the ready set
    //use the key in the application
}
```

LargerHttpd

- The LargerHttpd is a nonblocking web server that uses `SocketChannels` and a pool of threads to service requests.

LargerHttpd

- The LargerHttpd is a nonblocking web server that uses `SocketChannels` and a pool of threads to service requests.
- Example: LargerHttpd.java, HttpdConnection.java, ClientQueue.java.

LargerHttpd

- The LargerHttpd is a nonblocking web server that uses `SocketChannels` and a pool of threads to service requests.
- Example: LargerHttpd.java, HttpdConnection.java, ClientQueue.java.
- A single thread executes the main loop that accepts new connections and checks the readiness of existing client connections for reading or writing.

LargerHttpd

- The LargerHttpd is a nonblocking web server that uses `SocketChannels` and a pool of threads to service requests.
- Example: LargerHttpd.java, HttpdConnection.java, ClientQueue.java.
- A single thread executes the main loop that accepts new connections and checks the readiness of existing client connections for reading or writing.
- Whenever a client needs attention, it places the job in a queue where a thread from our thread pool waits to service it.
- Run it as follows (on one line!):

```
java -Djava.security.manager
     -Djava.security.policy=mysecurity.policy
      LargerHttpd <port> <maxthreads>
```

Exercises

- Modify the object server example to be single threaded.

Exercises

- Modify the object server example to be single threaded.
- Modify the object server example to use the Executor interface.

Exercises

- Modify the object server example to be single threaded.
- Modify the object server example to use the Executor interface.
- Create a EC2 virtual machine using AWS (Amazon Web Services) and run the object server on it. Connect with your laptop or a lab machine as a client.

Exercises

- Modify the object server example to be single threaded.
- Modify the object server example to use the Executor interface.
- Create a EC2 virtual machine using AWS (Amazon Web Services) and run the object server on it. Connect with your laptop or a lab machine as a client.
- Convert the object server example to send JSON objects instead of Java objects.

Exercises

- Modify the object server example to be single threaded.
- Modify the object server example to use the Executor interface.
- Create a EC2 virtual machine using AWS (Amazon Web Services) and run the object server on it. Connect with your laptop or a lab machine as a client.
- Convert the object server example to send JSON objects instead of Java objects.
- Write a Ping Pong program (two processes) using sockets and an object to represent the ping-pong ball.