IK1350 Protocols in Computer Networks/ Protokoll i datornätverk Spring 2008, Period 3

Module 6: TCP, HTTP, RPC, NFS, X

Lecture notes of G. Q. Maguire Jr.

KTH Information and Communication Technology

For use in conjunction with *TCP/IP Protocol Suite*, by Behrouz A. Forouzan, 3rd Edition, McGraw-Hill, 2006.

For this lecture: Chapter 12

Lecture 4: Outline

- TCP
- HTTP
- Web enabled devices
- RPC, XDR, and NFS
- X Window System, and
- some tools for looking at these protocols

Transport layer protocols

- User Datagram Protcol (UDP)
  - Connectionless unreliable service
- Transmission Control Protocol (TCP) <<< today's topic
  - Connection-oriented reliable stream service
- Stream Control Transmission Protocol (SCTP)
  - a modern transmission protocol with many facilities which the user can chose from

Transmission Control Protocol (TCP)

TCP provides a connection oriented, reliable, byte stream service[30].

- TCP utilizes full-duplex connections
  - Note: There are just **two** endpoints
- TCP applications write 8-bit bytes to a stream and read bytes from a stream
  - TCP decides how much data to send (not the application) - each unit is a **segment**
  - There are no records (or record makers) - just a stream of bytes ⇒ the receiver can't tell how much the sender wrote into the stream at any given time
- TCP provides reliability
  - Acknowledgements, timeouts, retransmission, …
- TCP provides flow control
- TCP trys to avoid causing congestion

Applications which use TCP

Lots of applications have been implemented on top of the TCP, such as:

- TELNET — provides a virtual terminal {emulation}
- FTP — used for file transfers
- SMTP — forwarding e-mail
- HTTP — transport data in the World Wide Web

Here we will focus on some features not covered in the courses: Telesys, gk and Data and computer communication.

TCP Header

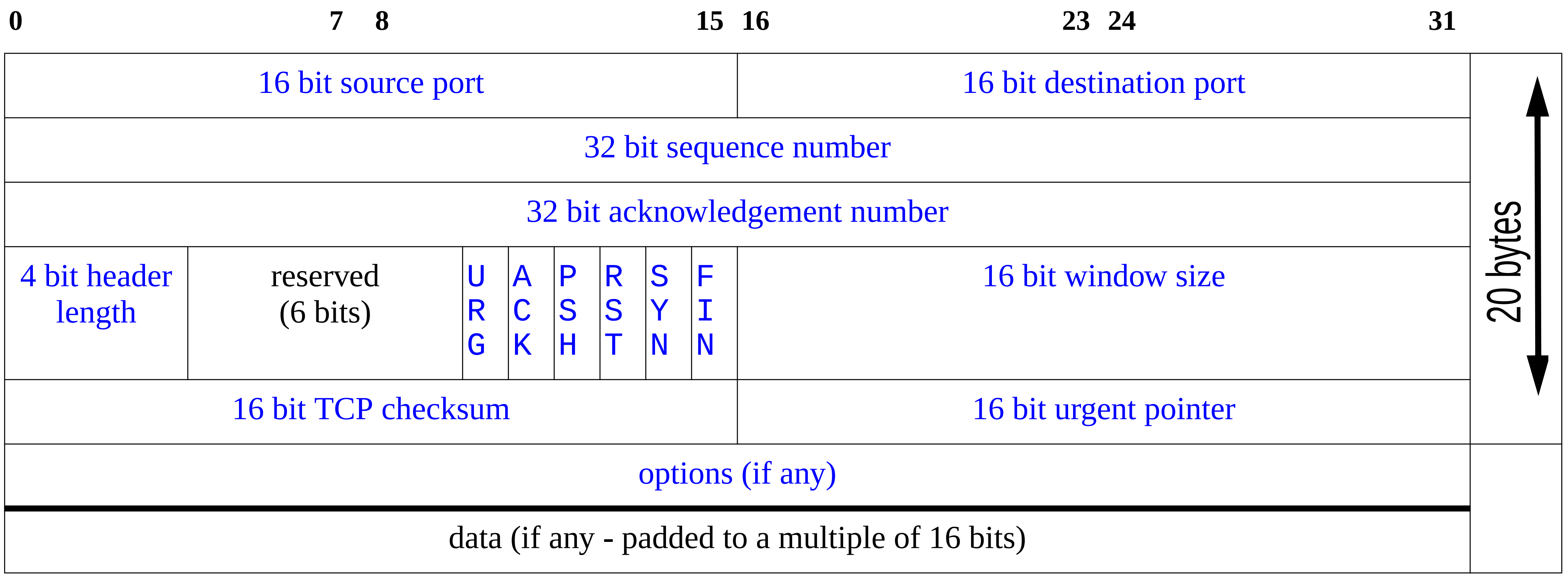


Figure 48: IP header (see Stevens, Vol. 1. inside cover or Forouzan figure 12.5 pg. 282)

Just as with UDP, TCP provides de/multiplexing via the 16 bit source and destination ports.

The 4 bit header length indicates the number of **4-byte words** in the TCP header

TCP header continued

Reliability is provided by the 32 bit sequence number which indicates the byte offset in a stream of the first byte in this segment and a 32 bit acknowledgement number which indicates the next byte which is **expected**.

- The initial sequence number (ISN) is a random 32 bit number.
- Note that the acknowledgement is **piggybacked** in each TCP segment
  - TCP maintains a timer for *each* segment. If an acknowledgement is not received before the timeout, then TCP retransmits the segment
  - When TCP receives data it sends an acknowledgement back to sender
- TCP applies an end-to-end checksum on its header and data
  - The checksum is mandatory - but otherwise similar to the UDP checksum
- TCP resequences data at the receiving side $\Rightarrow$ all the bytes are delivered in order to the receiving application
- TCP discards duplicate data at the receiving side

Urgent pointer - specifies that the stream data is offset and that the data field begins with "urgent data" which is to bypass the normal stream - for example ^C

Control field - indicates the purpose & contents of this segment:

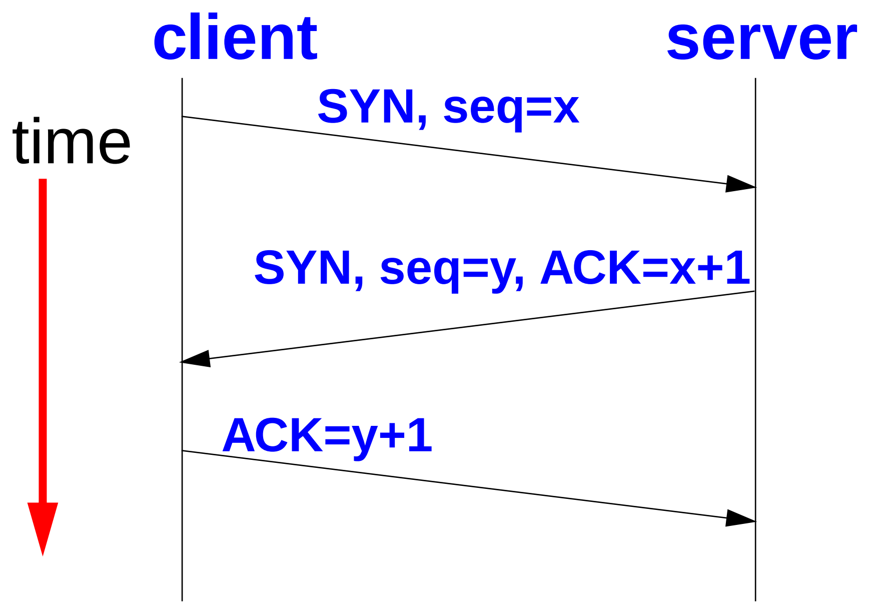
Flag	Description
URG	The urgent pointer is valid
ACK	The acknowledgement number is valid
PSH	Push the data, i.e., the receiver should immediately pass all the data to the application ⇒ emptying the receiver's buffer
RST	Connection must be rest
SYN	Synchronize the sequence numbers
FIN	Terminate the connection (from the **sender's** point of view)

We will see how these bits are used as we examine each of them later.

The window size (or more exactly the receive window size (rwnd)) - indicates how many bytes the receiver is prepared to receive (this number is relative to the acknowledgement number).

Options - as with UDP there can be up to 40 bytes of options (we will cover these later)

Connection establishment



3-way handshake:

- Guarantees both sides are ready to transfer data
- Allows both sides to agree on initial sequence numbers

See Forouzan figure 12.9 page 286

Initial sequence number (ISN) **must** be chosen so that each instance of a specific TCP connection between two end-points has a different ISN.

The entity initiating the connection is (normally) called the "client".

SYN Flooding Attack

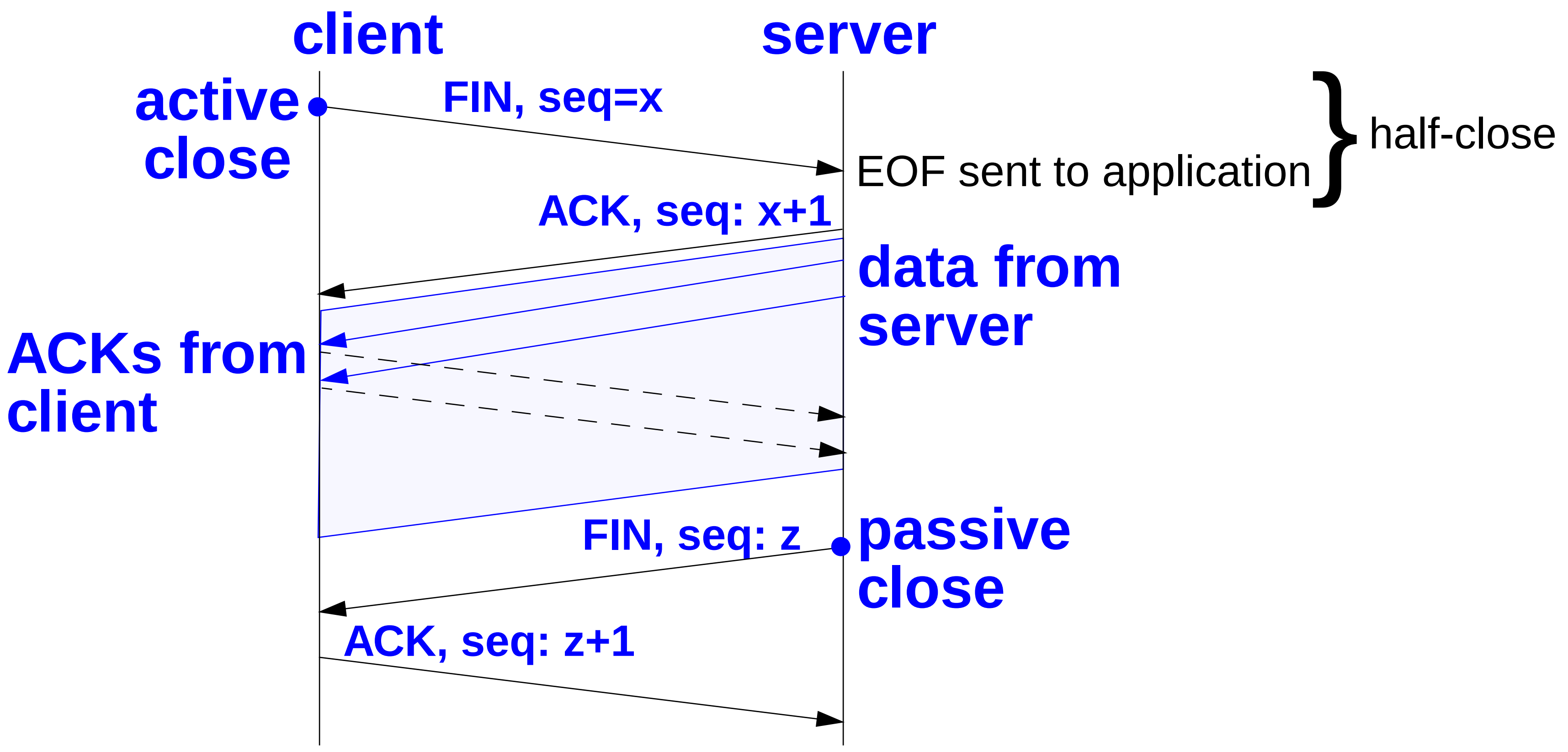
It is clear that if a malacioous user simply sends a lot of SYN sgements to a target machine (with faked source IP addresses) ⇒ this machine will spend a lot of resources to set up TCP connections which subsequently never occur.

As the number of TCP control blocks and other resources are finite

- legitimate connection requests can’t be answered
- the target machine might even crash

The result is to deny service, this is one of many Denial of Services (DoS) Attacks

Connection teardown



See Forouzan figure 12.12 page 291

- **Note:** it takes 4 segments to complete the close.
- Normally, the client performs an **active close** and the server performs **passive close.**

TCP options

These options are used to convey additional information to the destination or to align another options

- Single-byte Options
  - No operation
  - End of option
- Multiple-byte Options
  - Maximum segment size
  - Window scale factor
  - Timestamp

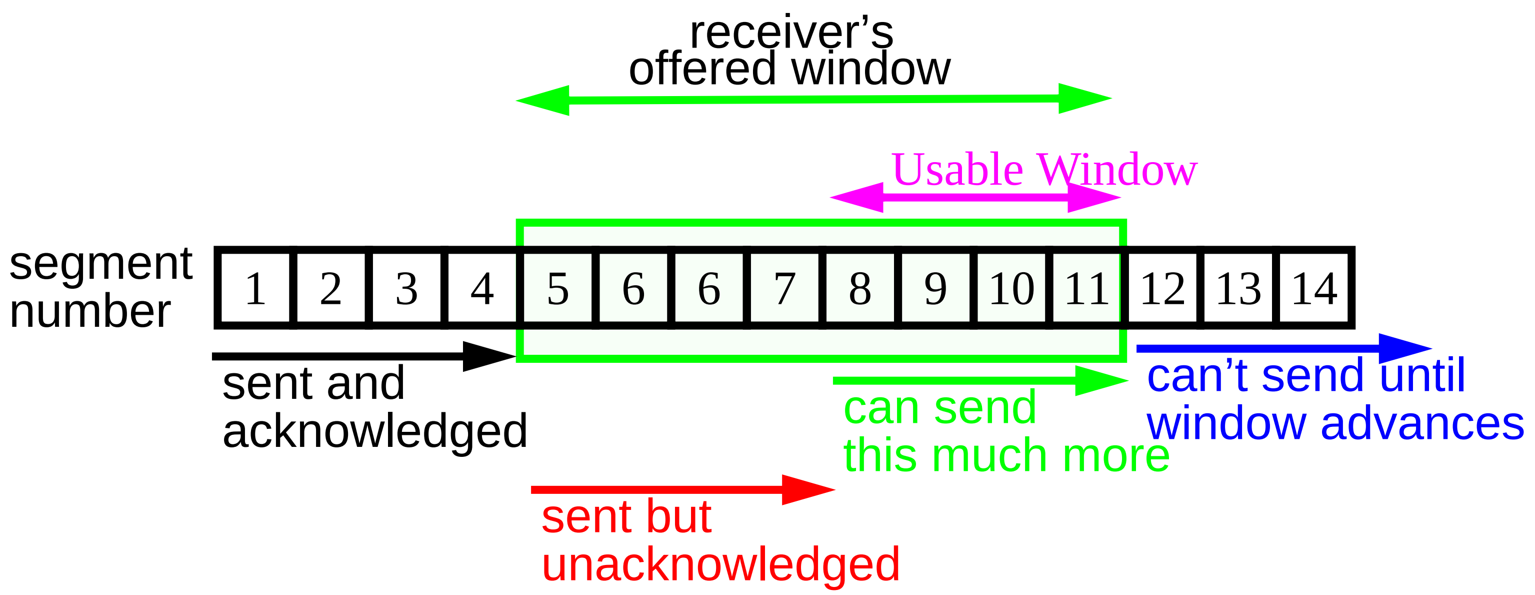
A TCP header can have up to 40 bytes of optional information.

Maximum Segment Size

- The Maximum Segment Size (MSS) is the largest amount of data TCP will send to the other side
- MSS can be announced in the options field of the TCP header during connection establishment
- If a MSS is not announced ⇒ a default value of 536 is assumed
- In general, the larger MSS is the better -- until fragmentation occurs
  - As when fragmentation occurs the overhead increases!

Sliding window Flow control

- receiver: **offered window** - acknowledges data sent and what it is prepared to receive
  - thus the sender can send an ACK, but with a offered window of 0
  - later the sender sends a **window update** with a non-zero offered window size
  - the receiver can increase or decrease this window size as it wants
- sender: **usable window** - how much data it is prepared to send immediately

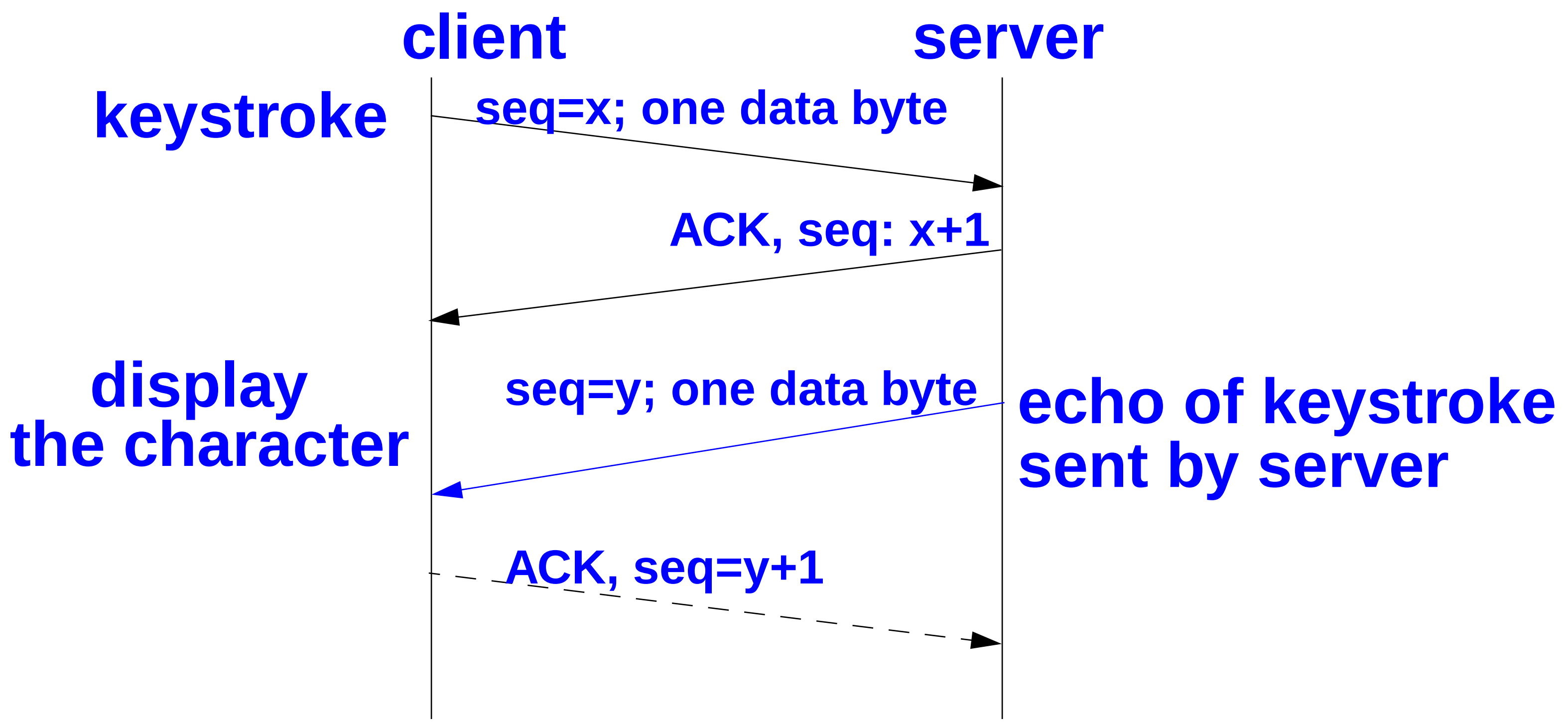


Window size

Increasing window size can improve performance - more recent systems have increased buffer size ranging from 4096 ... 16,384 bytes. The later produces ~40% increase in file transfer performance on an ethernet.

Socket API allows user to change the size of the send and receive buffers.

Flow for an rlogin session



See Forouzan figure 12.12 page 291

Thus **each** keystroke not only generates a byte of data for the remote application which has to be sent in a segments, but this will trigger an ACK along with an echo & its ACK ⇒ generates 3 more segments! All to send a single byte of user data!!!

Silly Window Syndrome

If receiver advertises a small window, then sender will send a small amount of data, which fills receivers window, … .

To prevent this behavior:

- sender does not transmit unless:
  - full-size segment can be sent OR
  - it can send at least 1/2 maximum sized window that the other has ever advertised
  - we can send everything we have and are not expecting an ACK or Nagle algorithm is disabled
- receiver must not advertise small segments
  - advertise **min**(a full-size segment, 1/2 the receive buffers space)
  - delayed ACKs

Nagle Algorithm

telnet/rlogin/... generate a packet (41 bytes) for each 1 byte of user data

- these small packets are called “tinygrams”
- not a problem on LANs
- adds to the congestion on WANs

Nagle Algorithm

- each TCP connection can have only one outstanding (i.e., unacknowledged) small segment (i.e., a tinygram)
- while waiting - additional data is accumulated and sent as one segment when the ACK arrives
- self-clock: the faster ACKs come, the more often data is sent
  - thus automatically on slow WANs fewer segments are sent

Round trip time on a typical ethernet is ~16ms (for a single byte to be sent, acknowledged, and echoed) - thus to generate data faster than this would require typing faster than 60 characters per second! Thus rarely will Nagle be invoked on a LAN.

Disabling the Nagle Algorithm

But sometimes we need to send a small message - without waiting (for example, handling a mouse event in the X Window System) - therefore we set:

- `TCP_NODELAY` on the socket
- Host Requirements RFC says that hosts **must** implement the Nagle algorithm, but there **must** be a way to disable it on individual connections.

Delayed acknowledgements

Rather than sending an ACK immediately, TCP waits ~200ms hoping that there will be data in the reverse direction - thus enabling a piggybacked ACK.

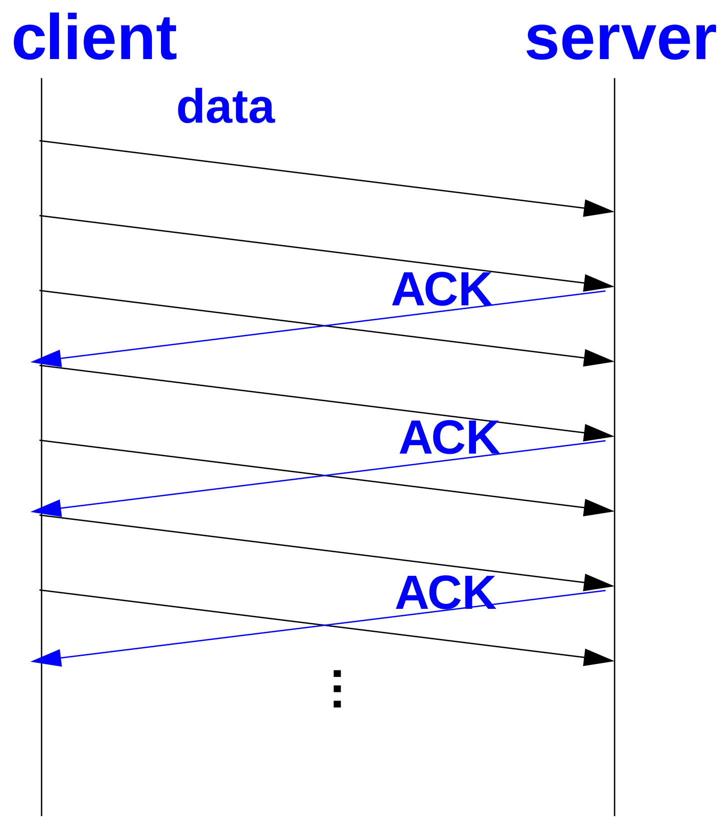
- Host Requirements RFC states the delays must be less than 500ms
- Implementations often use a periodic 200ms timer - rather than setting a timer specifically for computing this delay
  - similar to the periodic 500ms timer used for detecting timeouts

A complex protocol with several timers leads to complex behavior!

Resulting bulk data flow

Every segment cares a full MSS worth of data.

Typically an ACK every other segment.



Bandwidth-Delay Product

How large should the window be for optimal throughput?

Calculate the capacity of the pipe as:

- capacity(bits) = bandwidth(bits/sec) * RTT(sec)
- This is the size the receiver advertised window should have for optimal throughput.

Example:

T1 connection across the US:
```
    capacity = 1.544Mbit/s * 60ms = 11,580 bytes
```
Gigabit connection across the US:
```
    capacity = 1Gbit/s * 60ms = 7,500,000 bytes!
```

However, the window size field is only 16 bits ⇒ maximum value of 65535 bytes

For Long Fat Pipes we can use the window scale option to allow much larger window sizes.

Congestion Avoidance

So far we have assumed that the sender is only limited by the receiver's available buffer space. But if we inject lots of segments into a network - upto window size advertised by receiver

- works well if the hosts are on the same LAN
- if there are routers (i.e., queues) between them and if the traffic arrives faster than it can be forwarded, then either the packets have to be
  - buffered or
  - thrown away - we refer to this condition as congestion

Lost packets lead to retransmission by the sender

This adds even more packets to the network ⇒ **network collapse**

Therefore we need to be able to reduce the window size to avoid congestion.

Congestion Control

We introduce a Congestion Window

- Thus the sender’s window size will be determined both by the receiver and in reaction to congestion in the network

Sender maintains 2 window sizes:

- Receiver-advertised window (rwnd)
  - advertised window is flow control imposed by **receiver**
- Congestion window (CWND)
  - congestion window is flow control imposed by **sender**

Actual window size = min(rwnd, CWND)

To deal with congestion, sender uses several strategies:

- Slow start
- Additive increase of CWND
- Multiplicative decrease of CWND

Slow start

In 1989, Van Jacobson introduced **slow start** based on his analysis of actual traffic and the application of control theory. All TCP implementations are now required to support slow start.

- the rate at which new packets should be injected into the network is the rate at which acknowledgements are returned

- cwnd starts at number of bytes in one segment (as announced by other end) and increases exponentially with successfully received **cwnd** worth of data

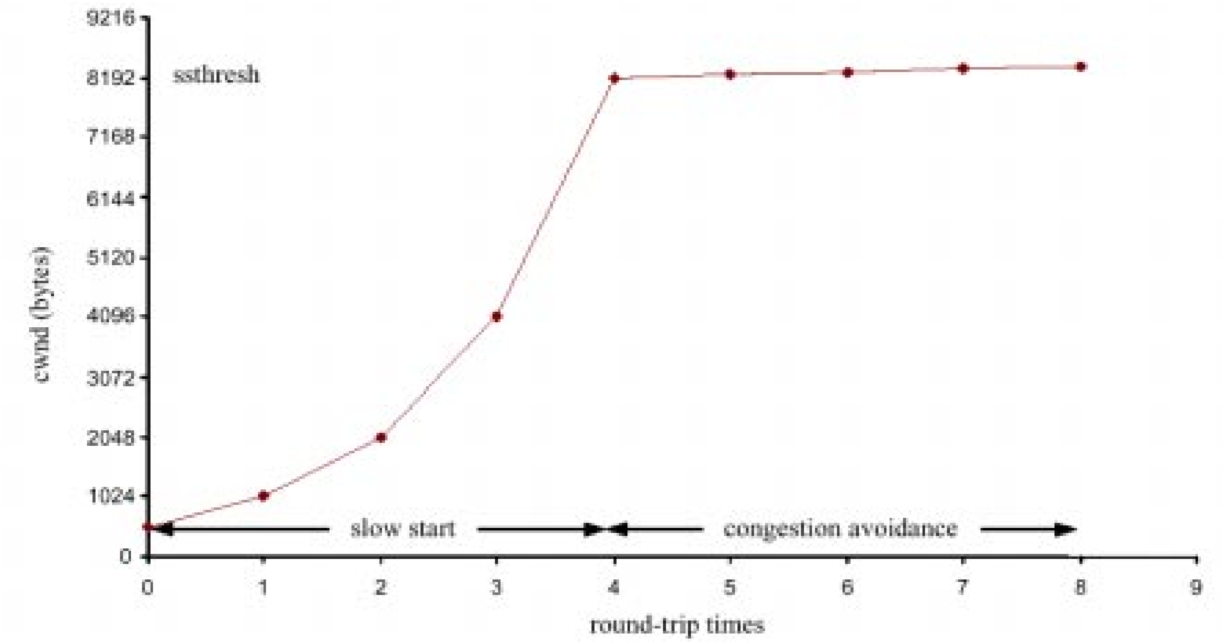


Figure 49: Graphical plot of congestion window (cwnd) as the connection goes from slow start to congestion avoidance behavior (figure from Mattias Ronquist, “TCP Reaction to Rapid Changes of the Link Characteristics due to Handover in a Mobile Environment”, MS Thesis, Royal Institute of Technology, Teleinformatics, August 4, 1999.)

Round-Trip Time Measurement

Fundamental to TCP's timeout and retransmission is the measurement (M) of the round-trip time (RTT). As the RTT changes TCP should modify its timeouts.

Originally TCP specificed:

$$R \leftarrow \alpha R + (1 - \alpha)M$$

α a smoothing factor, with a recommended value of 0.9

$$RTO = R\beta$$

β a delay variance factor, with a recommended value of 2

RTO == retransmission timout time

Van Jacobson found that this could not keep up with wide fluctuations in RTT, which leads to more retransmissions, when the network is already loaded! So he proposed tracking the variance in RTT and gave formulas which compute the RTO based on the **mean** and **variance** in RTT and can be easily calculated using integer arithmetic (see Stevens, Vol. 1, pg. 300 for details).

Karn’s algorithm

When a retransmisson occurs, RTO is backed off, the packet retransmitted with the new longer RTO, and an ACK is received.

But is it the original ACK or the new ACK?

- we don’t know, thus we don’t recalculate a new RTO until an ACK is received for a segment which is not retransmitted

Congestion Avoidance Algorithm

Slow start keeps increasing cwnd, but at some point we hit a limit due to intervening routers and packets start to be dropped.

The algorithm assumes that packet loss means congestion[1]. Signs of packet loss:

- timeout occuring
- receipt of duplicate ACKs

Introduce another variable for each connection: ssthresh == slow start threshold

when data is acknowledged we increase cwnd:

- if cwnd < ssthreshold we are doing slow start; increases continue until we are half way to where we hit congestion before
- else we are doing congestion avoidance; then increase by 1/cwnd + 1/8 of segment size each time an ACK is received

(See Stevens, Vol. 1, figure 21.8, pg. 311 for a plot of this behavior)

1. Note: if your losses come from other causes (such as bit errors on the link) it will still think it is due to congestion!

Van Jacobson’s Fast retransmit and Fast Recovery Algorithm

TCP is required to generate an immediate ACK (a duplicate ACK) when an out-of-order segment is received. This duplicate ACK should not be delayed. The purpose is to tell the sender that the segment arrived out of order and what segment number the receiver expects.

Cause:

- segments arriving out of order OR
- lost segment

If more than a small number (3) of duplicate ACKs are detected, assume that a segment has been lost; then retransmit the missing segment immediately (without waiting for a retransmission timeout) and perform congestion avoidance - but not slow start.

Why not slow start? Because the only way you could have gotten duplicate ACKs is if subsequent segments did arrive - which means that data is getting through.

Per-Route Metrics

Newer TCPs keeps smoothed RTT, smoothed mean deviation, and slow start threshold in the routing table.

When a connection is closed, if there was enough data exchanged (defined as 16 windows full) - then record the parameters

- i.e., 16 RTT samples $\Rightarrow$ smoothed RTT is accurate to ~5%

TCP Persist Timer

If the window size is 0 and an ACK is lost, then receiver is waiting for data and sender is waiting for a non-zero window!

To prevent deadlock, introduce a **persist timer** that causes sender to query the receiver periodically with **window probes** to find out if window size has increased.

Window probes sent every 60 seconds - TCP never gives up sending them.

TCP Keepalive Timer

No data flows across an idle TCP connection - connections can persist for days, months, etc. Even if intermediate routers and links go down the connection persists!

However, some implementations have a **keepalive timer.**

Host Requirements RFC gives 3 reasons **not** to use keepalive messages:

- can cause perfectly good connections to be dropped during transient failures
- they consume unnecessary bandwidth
- they produce additional packet charges (if you are on a net that charges per packet)

Host Requirements RFC says you can have a keepalive time but:

- it must not be enabled unless an application specifically asks
- the interval must be configurable, with a default of no less than 2 hrs.

TCP Performance

TCP’s path MTU discovery:

- use **min**(MTU of outgoing interface, MSS announced by other end)
- use per-route saved MTU
- once an initial segment size is chosen - all packets have don’t fragment bit set
- if you get an ICMP “Can’t fragment” message - recompute Path MTU.
- periodically check for possibility of using a larger MTU (RFC 1191 recommends 10 minute intervals)

Long Fat Pipes

Networks with large bandwidth-delay products are called **Long Fat Networks (LFNs)** - pronounced “elefants”.

TCP running over a LFN is a **Long Fat Pipe.**

- Window Scale option - to avoid 16 bit window size limit
- Timestamp option - putting a time stamp in each segment allows better computation of RTT
- Protection Against Wrapped Sequence Numbers (PAWS) - with large windows you could have sequence number wrap around and not know which instance of a given sequence number is the correct one; solved by using timestamps (which must simply be monotonic)
- T/TCP - TCP extension for Transactions; to avoid the three way handshake on connection setup and shorten the TIME_WAIT state. (for details of T/TCP see Stevens, Vol.3)

Measuring TCP Performance

Measured performance:

- Performance on Ethernets at ~90% of theoretical value (using workstations)
- TCP over FDDI at 80-98% of theoretical value
- TCP (between two Crays) at 781 Mbits/sec over a 800Mbit/sec HIPPI channel
- TCP at 907Mbits/sec on a loopback interface of a Cray Y-MP.

Practical limits:

- can’t run faster than the slowest link
- can’t go faster than the memory bandwidth of the slowest machine (since you have to touch the data at least once)
- you can’t go faster than the window size offered by the receiver divided by the round trip time (comes from the calculation of the bandwidth delay product)
  - thus with the maximum window scale factor (14) $\Rightarrow$ window size of 1.073 Gbits; just divide by RTT to find the maximum bandwidth

Example of TCP behavior [1]

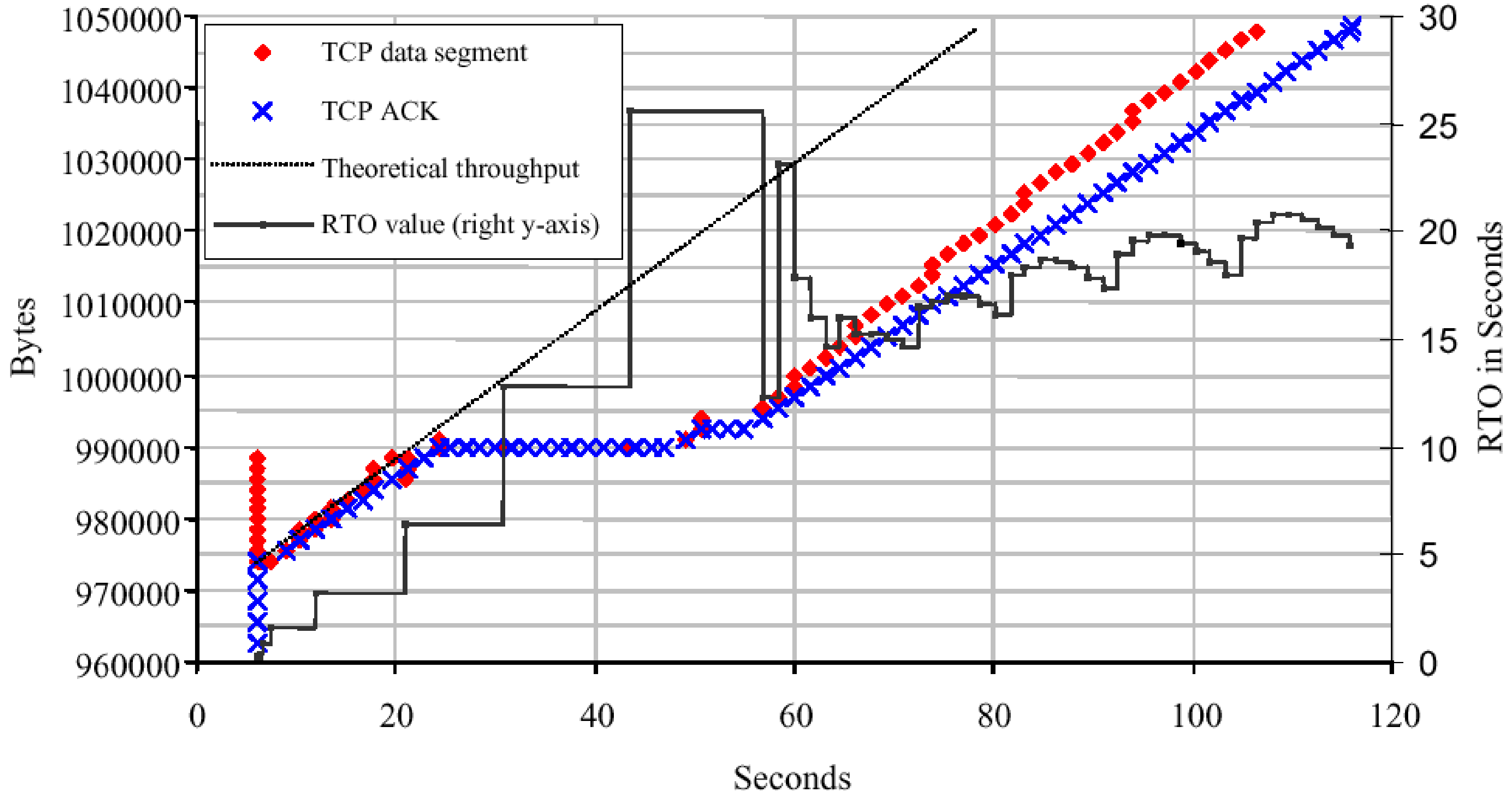


Figure 7-3: TCP segments and ACKs collected at the correspondent host after the handover. The RTO values and the theoretical throughput of the PPP link are also included.

1. Figure 7-3, from Mattias Ronquist, "TCP Reaction to Rapid Changes of the Link Characteristics due to Handover in a Mobile Environment", MS Thesis, Royal Institute of Technology, Teleinformatics, August 4, 1999, p.38.

TCP servers

Stevens, Vol. 1, pp. 254-260 discusses how to design a TCP server, which is similar to list of features discussed for UDP server, but now it is incoming connection requests which are queued rather than UDP datagrams

- note that incoming requests for connections which exceed the queue - are silently ignored - it is up to the sender to time out it active open
- this limited queuing has been one of the targets of denial of service attacks
  - TCP SYN Attack - see *http://cio.cisco.com/warp/public/707/4.html*
  - Increase size of the SYN_RCVD queue (kernel variable somaxconn limits the maximum backlog on a listen socket - backlog is the sum of both the SYN_RCVD and accept queues) and decrease the time you will wait for an ACK in response to your SYN_ACK
  - for a nice HTTP server example, see *http://www.cs.rice.edu/CS/Systems/Web-measurement/paper/node3.html*

Hypertext Transfer Protocol (HTTP)

This protocol is the basis for the World Wide Web (WWW).

Uses TCP connections. HTTP traffic growing at a very high rate - partly due to popularity and partly due to the fact that it can easily include text, pictures, movies, … .

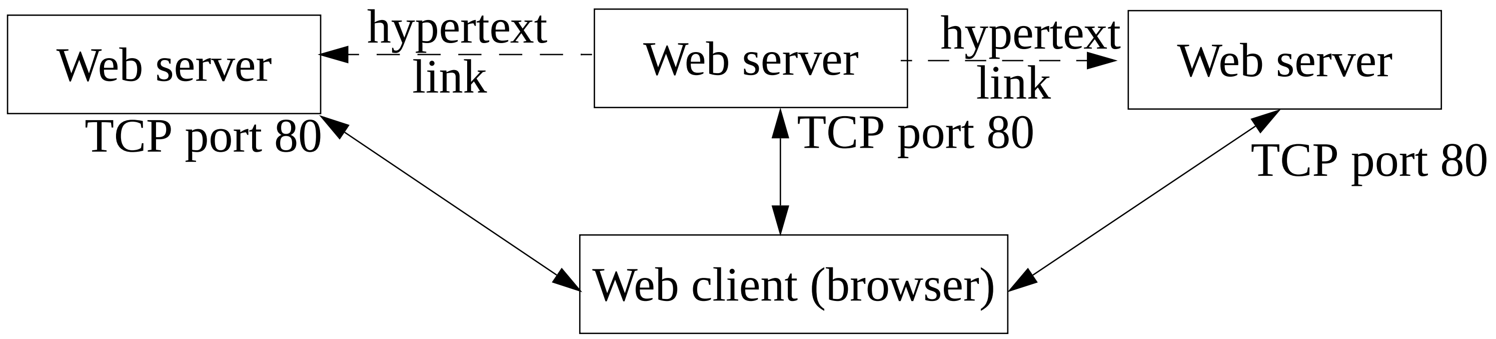


Figure 50: Organization of a Web client-server
(see Stevens, Vol. 3, figure 15.1, pg. 210)

HTTP described by an Internet Draft in 1993; replaced with RFC 1945, “*Hypertext Transfer Protocol -- HTTP/1.0*”, May 1996; RFC 2068, “*Hypertext Transfer Protocol -- HTTP/1.1*”, January 1997; replaced by RFC 2616, June 1999, RFC 2817 “*Upgrading to TLS Within HTTP/1.1*”, May 2000.

HTTP Prococol

2 message types:

- request

HTTP 1.0 request

request line
headers (0 or more)
<blank_line>
body (only for a POST request)

- response

HTTP/1.0 response

status line
headers (0 or more)
<blank_line>
body

HTTP Requests

request-line == request request-URI HTTP-version

Three requests:

- GET - returns information identified by request URI
- HEAD - similar to GET but only returns header information
- POST - sends a body with a request; used for posting e-mail, news, sending a fillin form, etc.

Universal Resource Idendifiers (URIs) - described in RFC 1630, URLs in RFC 1738 and RFC 1808.

status-line == HTTP-version response-code response-phrase

HTTP Header fields

HTTP Header names (See Stevens, figure 13.3, Vol. 3, pg. 166)

Header name	Request?	Response?	Body?
Allow			
Authorization			
Content-Encoding			
Content-Length			
Content-Type			
Date			
Expires			
From			
If-Modified-Since			
Last-Modified			
Location			
MIME-Version			
Pragma			
Referer			
Server			
User-Agent			
WWW-Authenticate			

HTTP Response Codes

HTTP 3-digit response code (See Stevens, figure 13.4, Vol. 3, pg. 167)

Response	**Description**
1yz	Informational. Not currently used
	Success
200	OK, request succeeded.
201	OK, new resource created (POST command)
202	Request accepted but processing not completed
204	OK, but no content to return
	Redirection; further action needs to be taken by user agent
301	Requested resource has been assigned a new permanent URL
302	Requested resource resides temporarily under a different URL
304	Document has not been modified (conditional GET)
	Client error
400	Bad request
401	Unauthorized; request requires user authentication
403	Forbidden for unspecified reason
404	Not found
	Server error
500	Internal server error
501	Not implemented
502	Bad gateway; invalid response from gateway or upstream server
503	Service temporarily unavailable

Client Caching

Client can cache HTTP documents along with the date and time the document was fetched.

If the document is cached, then the If-Modified-Since header can be sent to check if the document has changed since the copy was cached - thus saving a transfer - but costing a round trip time and some processing time. This is called a conditional GET.

Server Redirect

Response code 302, along with a new location of the request-URI.

Multiple simultaneous connections to server

GET of a page with multiple objects on it (such as GIF images) - one new connection for each object, all but the first can occur in parallel!

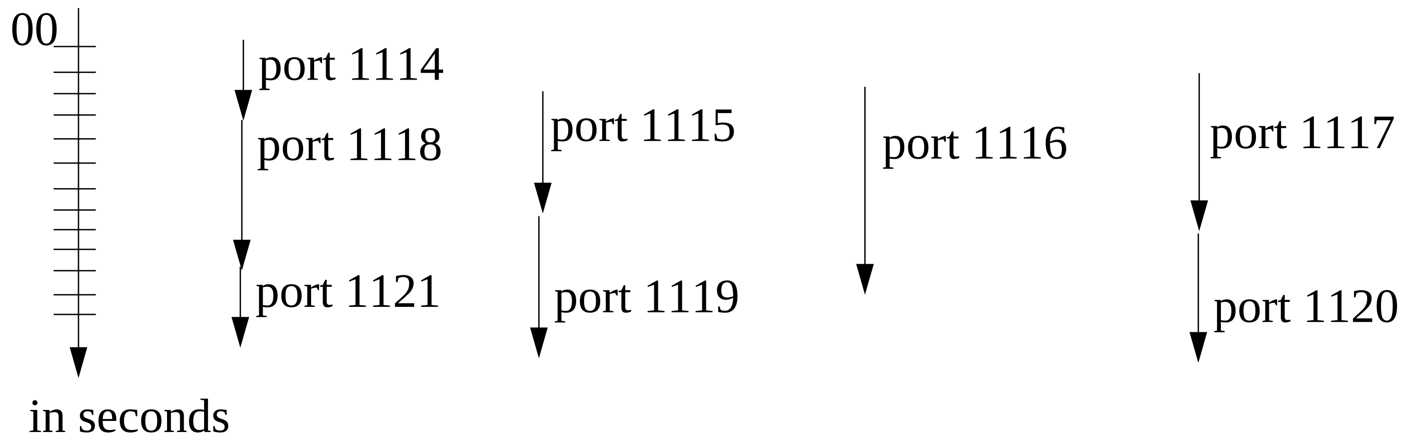


Figure 51: Timeline of eight TCP connection for a home page and seven GIF images (see Stevens, Vol. 3, figure 113.5, pg. 171)

Note that the port 1115, 1116, and 1117 requests start before 1114 terminates, Netscape can initiate 3 non-blocking connects after reading the end-of-file but before closing the first connection.

Decrease in total time to produce a response:

(from Stevens, figure 13.6, Vol. 3, pg. 171)

Simultaneous connections	**Total time (seconds)**
1	14.5
2	11.4
3	10.5
4	10.2
5	10.2
6	10.2
7	10.2

Why no improvement beyond 4?

- program has an implementation limit of 4, even if you specify more!
- gains beyond 4 are probably small (given the small difference between 3 and 4) - but Steven’s has not checked!

Problems with multiple connections

Such multiple connection have problems:

- Unfair to other protocols (such as FTP) which are using one connection at a time to fetch multiple files
- Congestion information is not passed to the other connections
- can more easily overflow the server's incomplete connection queue which can lead to large delays as the host retransmits SYNs.
  - ♦ In fact it looks like a denial of service attack -- which is trying to flood the host!

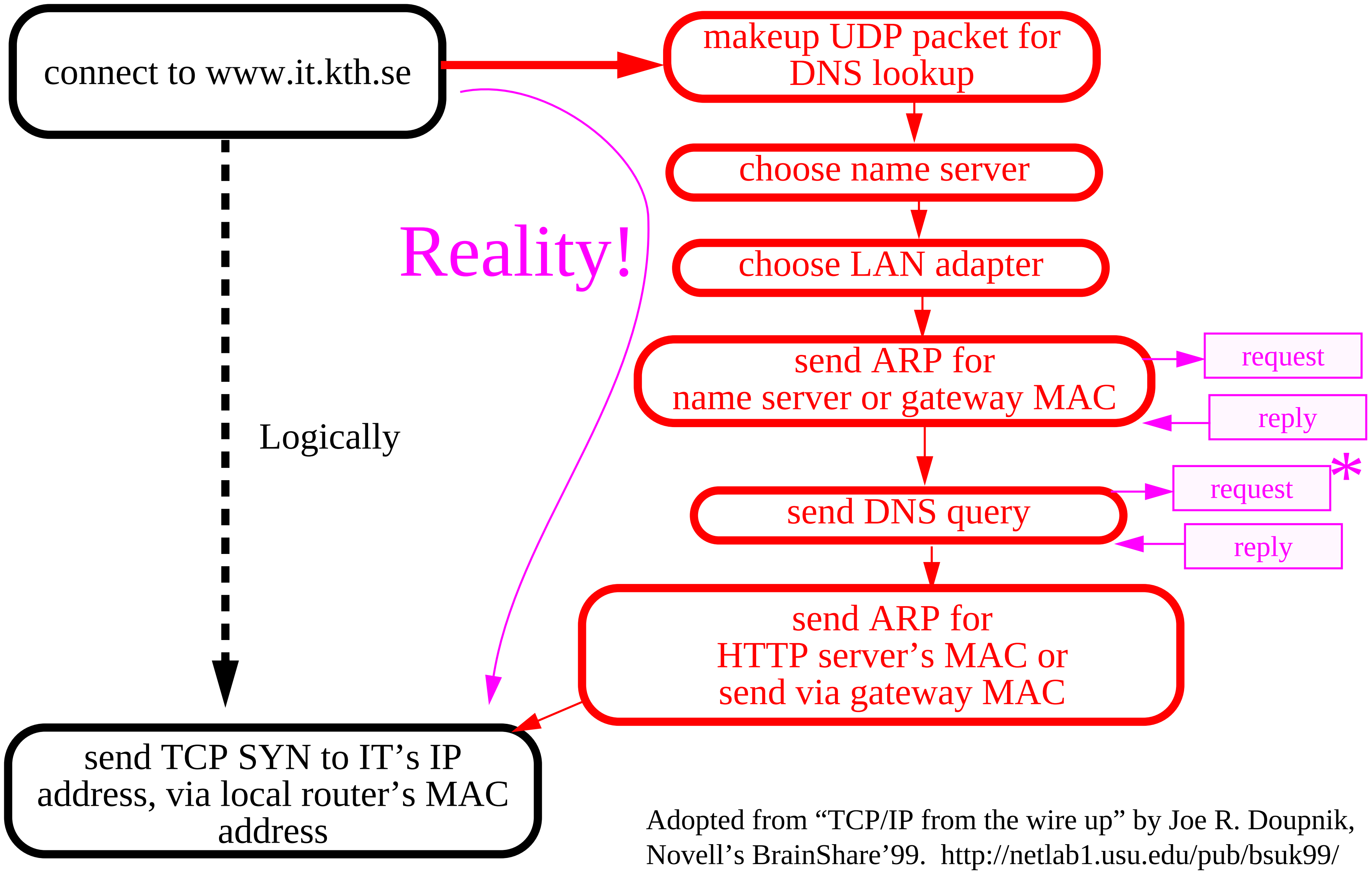
HTTP Statistics

Statistics for individual HTTP connections (see Stevens, figure 13.7, Vol. 3, pg. 172)

	Median	**Mean**
client bytes/connection	224	266
server bytes/connection	3,093	7,900
connection duration (seconds)	3.4	22.3

Mean is often skewed by very large transfers.

What happens when you make an HTTP request



connect to www.it.kth.se
makeup UDP packet for DNS lookup
choose name server
choose LAN adapter
send ARP for name server or gateway MAC
request
reply
send DNS query
request
*
reply
send ARP for HTTP server's MAC or send via gateway MAC
Reality!
Logically
send TCP SYN to IT's IP address, via local router's MAC address

Adopted from "TCP/IP from the wire up" by Joe R. Doupnik, Novell's BrainShare'99. http://netlab1.usu.edu/pub/bsuk99/

HTTP Performance Problems

HTTP opens one connection for **each** document.

- Each such connection involves slow start - which adds to the delay
- Each connection is normally closed by the HTTP server - which has to wait TIME_WAIT, thus lots of control blocks are waiting in the server.

Proposed changes:

- have client and server keep a TCP connection open {this requires that the size of the response (Content-Length) be generated}
  - requires a change in client and server
  - new header Pragma: hold-connection
- GETALL - causes server to return document and all in-lined images in a single response
- GETLIST - similar to a client issuing a series of GETs
- HTTP-NG (aka HTTP/1.1) - a single TCP connection with multiple sessions {it is perhaps the first TCP/IP session protocol}
  - HTTP/1.1 also has another feature - the server knows what hostname was in the request, thus a single server at a single IP address can be the HTTP server under many “names” - hence providing “Web hotel” services for many firms _but_ only using a single IP address.

HTTP performance

Joe Touch, John Heidemann, and Katia Obraczka, “Analysis of HTTP Performance”,USC/Information Sciences Institute, August 16, 1996, Initial Release, V1.2 -- *http://www.isi.edu/lsam/publications/http-perf/*

John Heidemann, Katia Obraczka, and Joe Touch, “Modeling the Performance of HTTP Over Several Transport Protocols”, IEEE/ACM Transactions on Networking 5(5), October 1997. November, 1996.
http://www.isi.edu/~johnh/PAPERS/Heidemann96a.html

Simon E Spero, “Analysis of HTTP Performance problems”
http://sunsite.unc.edu/mdma-release/http-prob.html This is a nice introduction to HTTP performance.

John Heidemann, “Performance Interactions Between P-HTTP and TCP Implementations”. ACM Computer Communication Review, 27 2, 65-73, April, 1997. *http://www.isi.edu/~johnh/PAPERS/Heidemann97a.html*

Web Enabled Devices

emWare - thin client (30 bytes of RAM, 750 bytes of ROM) - for a very thin client

URL: *http://www.emware.com/*

Splits the web server into a very tiny server on the device and more processing (via applets) in the desktop system (where the WEB browser is running).

Axis Communications AB - *http://www.axis.com* - produces many web enabled devices - from thin clients to “cameras” running an embedded Linux

Network File System (NFS)

NFS is based on Sun’s Remote Procedure Call (RPC) - RFC 1831

- from the caller’s point of view it looks much like a local procedure call
- from the callee’s point of view it seems much like a local procedure call
- Request-reply protocol
- UDP or TCP transport
- Standardized data representation - RPC encodes its data using the eXternal Data Representation (XDR) protocol, RFC 1832
- Authentication {for example, for NFS operations, authentication usually based on relaying UNIX user and group IDs to the file server for permission checking}

Remote Procedure Call (RPC)

Two versions:

- using Sockets API and works with TCP and UDP
- using TLI API TI-RPC (Transport Independent) and works with any transport layer provided by the kernel

Format of an RPC call message as a UDP datagram

IP Header	20 bytes
UDP Header	8
transaction ID (XID)	4
call (0)	4
RPC version (2)	4
program number	4
version number	4
procedure number	4
credentials	upto 400 bytes
verifier	upto 400 bytes
procedure parameters	N

- XID set by client and returned by server (client uses it to match requests and replies)

- program number, program version, procedure number identifies the procedue to be called
- credentials identify the client - sometimes the user ID and group ID
- verifier - used with secure RPC (to identify the server); uses DES encryption

l

Format of an RPC reply message as a UDP datagram

IP Header	20 bytes
UDP Header	8
transaction ID (XID)	4
call (1)	4
status (0=accepted)	4
verifier	upto 400 bytes
procedure results	N

External Data Representation (XDR)

used to encode value in RPC messages - see RFC 1014

Port Mapper

RPC server programs use ephemeral ports - thus we need a well known port to be able to find them

Servers register themselves with a registrar - the **port mapper**
(called rpcbind in SVR4 and other systems using TI-RPC)

Port mapper is at well know port: 111/UDP and 111/TCP

The port mapper is an RPC server with program number 100000, version 2, a TCP port of 111, a UDP port of 111.

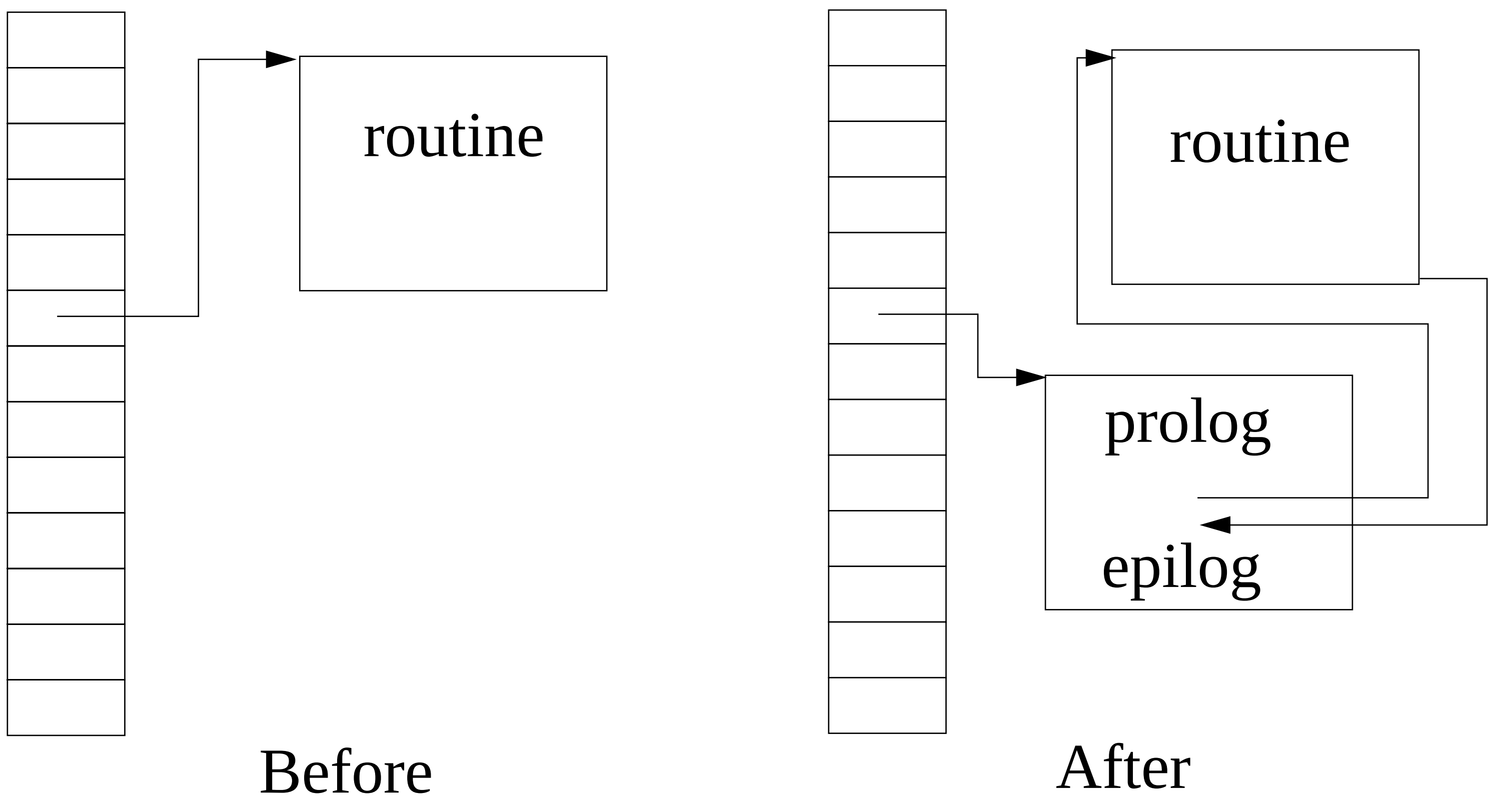
Servers register themselves with RPC calls and clients query with RPC calls:

- PMAPPROC_SET - register an entry
- PMAPPROC_UNSET - unregister an entry
- PMAPPROC_GETPORT - get the port number of a given instance
- PMAPPROC_DUMP - returns all entries (used by “rpcinfo -p”)

NFSspy

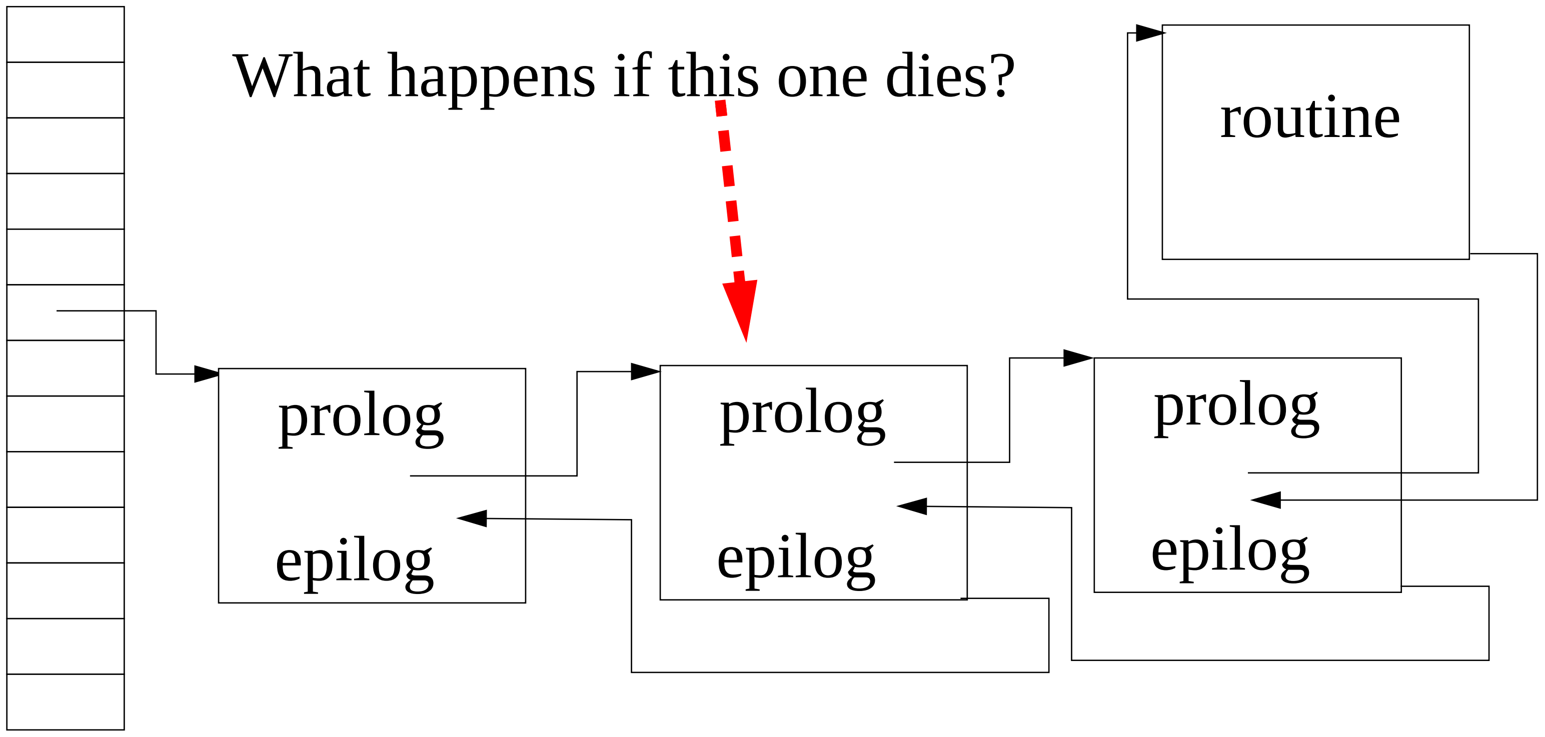
Insert a new pointer in place of the RPC we want to snoop

Embed this earlier code in our code:



NFSspy problem

Imagine several students each insert a new pointer in place of the RPC they want to snoop:



The routine is no longer callable by anyone!

nfsspy

Initial implementations were written by Seth Robertson, Jon Helfman, Larry Ruedisueli, Don Shugard, and other students for a project assignment in my course on Computer Networks at Columbia Univ. in 1989. There is a report about one implementation by Jon Helfman, Larry Ruedisueli, and Don Shugard, "Nfspy: A System for Exploring the Network File System", AT&T Bell Laboratories, 11229-890517-07TM.

See also “NFS Tracing By Passive Network Monitoring” by Matt Blaze, ~1992, *http://www.funet.fi/pub/unix/security/docs/papers/nfsspy.ps.gz*

- Matt’s program builds upon an rpcspy program and this feeds packets to his nfstrace program and other scripts.

Seth Robertson’s version even inverted the file handles to show the actual file names.

NFS protocol, version 2 (RFC 1094)

- provides transparent file access
- client server application built on RPC

Generally, NFS server at 2049/UDP - but it can be at different ports

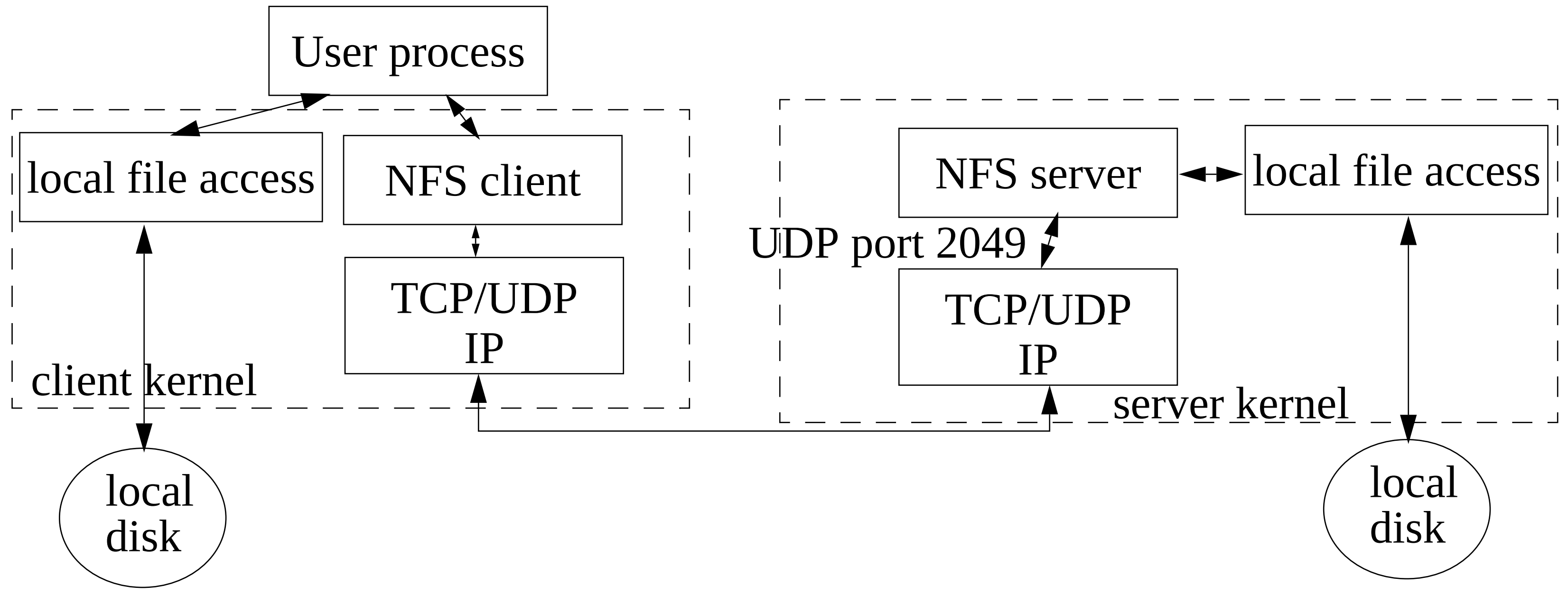


Figure 52: NFS client and NFS server (see Stevens, Vol. 1, figure 29.3, pg. 468)

Most NFS servers are multithreaded - so that multiple requests can be in process at one time. If the server kernel does not support mutlithreading then multiple server processes (“nfsd”) are used.

Often there are multiple NFS clients (“biod”) running on the client machine - each processes one call and waits inside the kernel for the reply.

NFS consists of more than just the NFS protocol

Various RPC programs used with NFS (see Stevens, Vol. 1, pg. 469)

Application	**program number**	**version numbers**	**Number of procedures**
port mapper	100000	2	4
NFS	100003	2	15
mount	100005	1	5
lock manager	100021	1,2,3	19
status monitor	100024	1	6

lock manager and status monitor allow locking of portions of files

NFS File Handles

To reference a file via NFS we need a **file handle,** an opaque object used to reference a file or directory on the server.

File handle is created by the server - upon an lookup; subsequent client requests just simply pass this file handle to the appropriate procedure (they never look at the contents of this object - hence it is opaque).

- in version 2, a file handle is 32 bytes
- in version 2, a file handle is 64 bytes

UNIX systems generally encode the filesystem ID (major and minor dev numbers), the i-node number, and an i-node generation number into the file handle.

NFS Mount protocol

Server can check IP address of client, when it gets a mount command from a client to see if this client is allowed to mount the given filesystem; Mount daemon returns the file handle of the given filesystem.

NFS Procedures

Procedure	**Description**
NFSPROC_GETATTR	return the attributes of a file
NFSPROC_SETATTR	set the attributes of a file
NFSPROC_STATFS	return the status of a filesystem
NFSPROC_LOOKUP	lookup a file - returns a file handle
NFSPROC_READ	read from a file, starting at specified offset for n bytes (upto 8192 bytes)
NFSPROC_WRITE	Write to a file, starting at specified offset for n bytes (upto 8192 bytes) Writes are synchronous - i.e., server responds OK when file is actually written to disk (this can often be changed as an option at mount time - but you can get into trouble)
NFSPROC_CREATE	Create a file
NFSPROC_REMOVE	Delete a file
NFSPROC_RENAME	Rename a file
NFSPROC_LINK	make a hard link to a file
NFSPROC_SYMLINK	make a symbolic link to a file
NFSPROC_READLINK	return the name of the file to which the symbolic link points
NFSPROC_MKDIR	create a directory
NFSPROC_RMDIR	delete a directory
NFSPROC_READDIR	read a directory

NFS over TCP

Provided by some vendors for use over WANs.

- All applications on a given client share the same TCP connection.
- Both client and server set TCP keepalive timers
- If client detects that server has crashed or been rebooted, it tries to reconnect to the server
- if the client crashes,the client gets a new connection, the keepalive timer will terminate the half-open former connection

NFS Statelessness

NFS is designed to be stateless

- the server does not keep track of what clients are accessing which files
- there are no open or close procedures; just LOOKUP
- being stateless simplifies server crash recovery
- clients don’t know if the server crashes
- only the client maintains state

Most procedures (GETATTR, STATFS, LOOKUP, READ, WRITE, READIR) are idempotent (i.e., can be executed more than once by the server with the same result).

Some (CREATE, REMOVE, RENAME, SYMLINK, MKDIR, RMDIR) are not. SETATTR is idempotent unless it is truncating a file.

To handle non-idempotent requests - most servers use recent-reply cache, checking their cache to see if they have already performed the operation and simply return the same value (as before).

X Window System

- Client-server application that lets multiple clients share a bit-mapped display.
- One server manages the display, keyboard, mouse, …
- X requires a reliable bidirectional bitstream protocol (such as TCP).
- The server does a passive open on port 6000+n, where n is the display number (usually 0)
- X can also use UNIX domain sockets (with the name /tmp/.X11-unix/Xn, where n is the display number)
- > 150 different messages in the X protocol (for details see Nye, 1992)

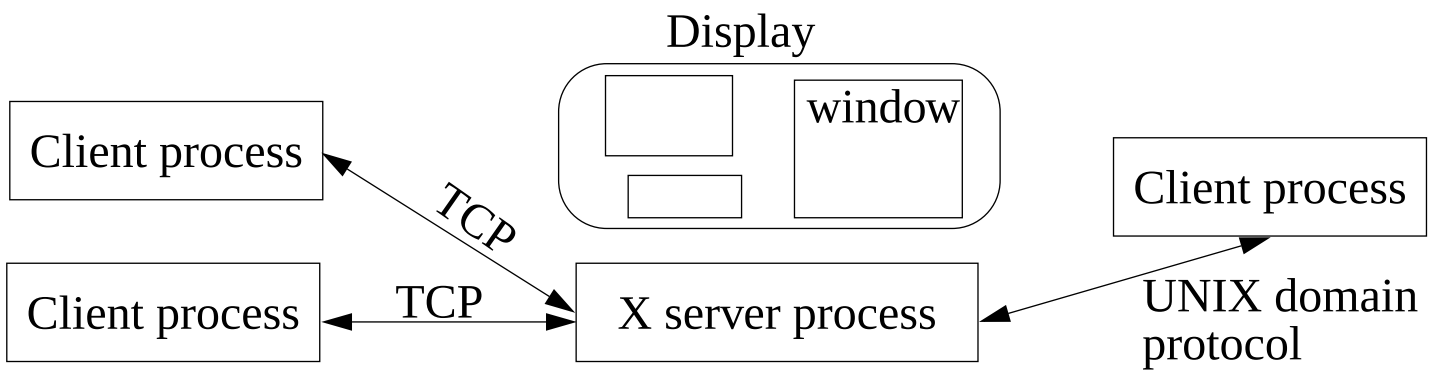


Figure 53: Clients using a X server to access one display

- All clients (even those on different hosts) communicate with the same server.
- Lots of data can be exchanged between client and server
  - xclock - send date and time once per second
  - Xterm - send each key stroke (a 32 byte X message $\Rightarrow$ 72 bytes with IP and TCP headers)
  - some applications read and write entire 32 bit per pixel images in cine mode from/to a window!

Low Bandwidth X

X was optimized for use across LANs.

For use across low speed links - various techniques are used:

- caching
- sending differences from previous packets
- compression, …

Xscope

Interpose a process between the X server and X client to watch traffic.

For example, xscope could be run as if it were display “1”, while passing traffic to and from display “0”. See Stevens, Vol.1, pp. 488-489 for more details (or try running it!)

J.L. Peterson. XSCOPE: A Debugging and Performance Tool for X11. Proceedings of the IFIP 11th World Computer Congress, September, 1989, pp. 49-54.

See also XMON - An interactive X protocol monitor

Both are available from: *ftp://ftp.x.org/pub/R5/*

Additional tools for watching TCP

Program	Description
IPerf	Measure bandwidth availabity using a client and server. Determines total bandwidth, delay jitter, loss, determine MTU, support TCP window size, …
Pathchar	Determine per hop network path characteristics (bandwidth, propagation delay, queue time and drop rate. It utilizes a series of packets with random payload sizes over a defined period of time to each hop in a path.
Pchar	Updated version of Pathchar -- by Bruce Mah
Netlogger	NetLogger includes tools for generating precision event logs that can be used fpr detailed end-to-end application & system level monitoring, and tools for visualizing log data to view the state of a distributed system in real time.
Treno	Measure single stream bulk transfer capacity. TReno doesn't actually use TCP slow start, but instead emulates it. It actually sends UDP packets to unused ports and uses the returned error meassages to determine the packet timing.
Mping	Measure queuing properties during heavy congestion
tdg	produce graphs of TCP connections from tcpdump files, suitable for use with xgraph. A perl script which produces time-sequence plots from tcpdump files.

Program	Description
tcptrace	parses raw tcpdump files to extract information and xplot files.
xplot	generate graphs from plot data in X Windows.
testrig	automated connection diagnosis tool; generates a test flow & display a time sequence plot based on that flow.
aspath	determine traffic usage by AS path.

Transaction TCP (T/TCP)

Piggyback a query in the TCP open - so that you don't have to wait a long time for sending a query / response

Summary

This lecture we have discussed:

- TCP
- HTTP
- Web enabled devices
- RPC, XDR, and NFS
- X Window System, and
- some tools for looking at these protocols

References

[30] Information Sciences Institute, University of Southern California, Transmission Control Protocol, IETF, RFC 793, September 1981