ADVANCED

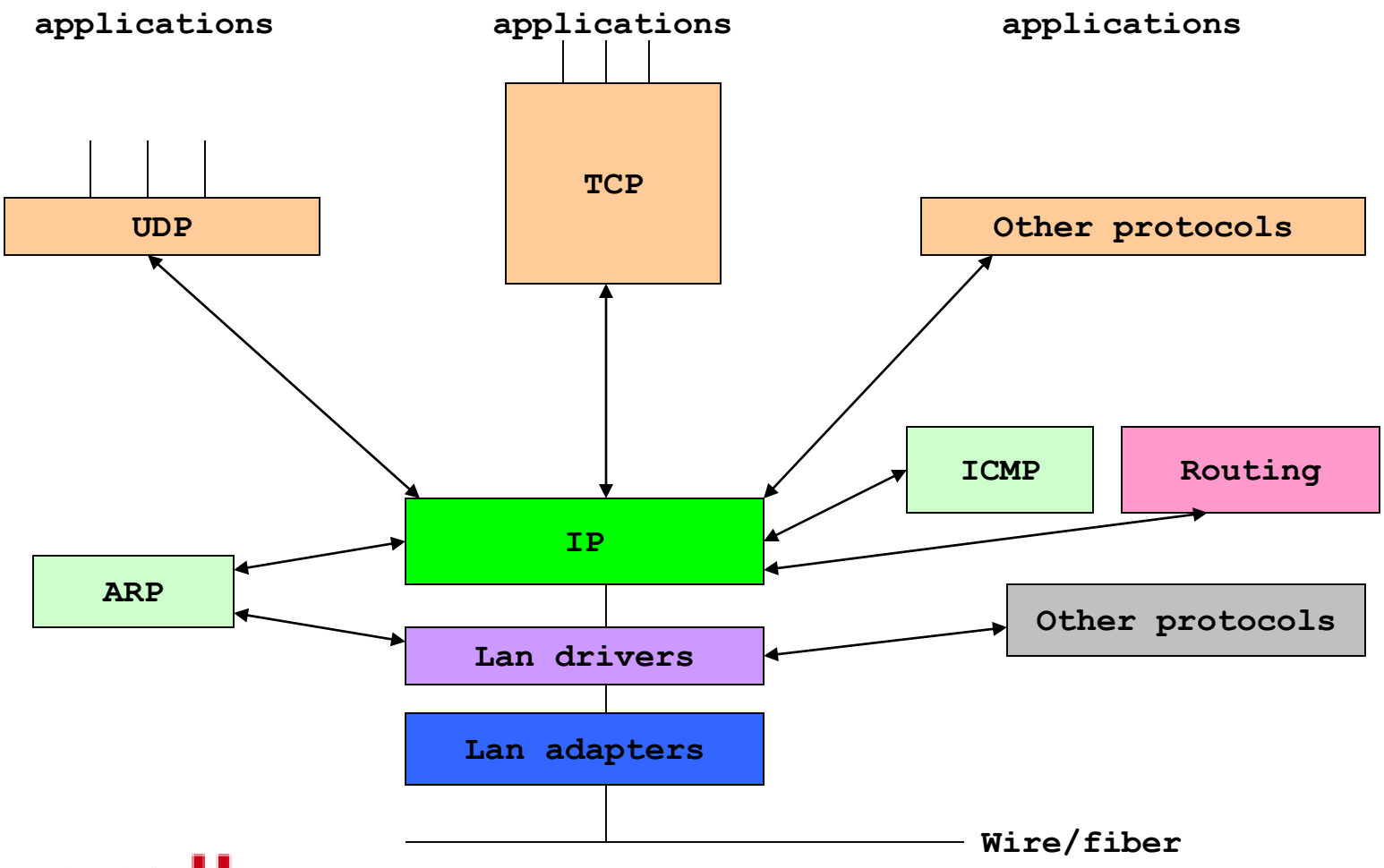
TCP/IP from the wire up

Joe R. Doupnik
Utah State University
jrd@cc.usu.edu

Novell

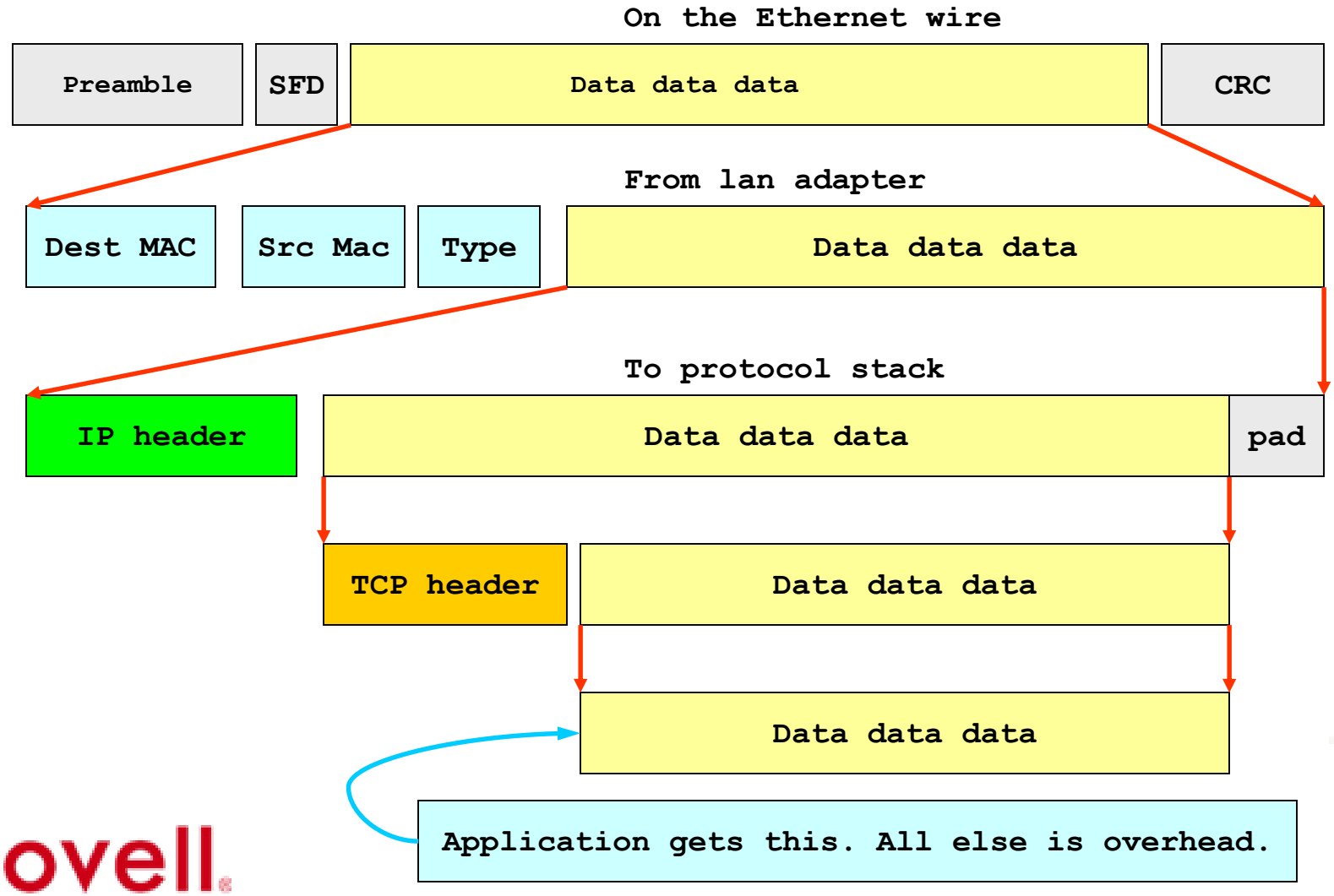
BRAINSHARE '99

TCP/IP stack layout



Novell

From the wire into the application



Novell

Internet Protocol (IP)

- Transportation services
- Understands IP addresses, elementary routing
- Adds IP header to route traffic with IP addresses
- Performs packet fragmentation and reassembly
- Has Time To Live for routing

IP, cont'd

- No ACKs; it is send and forget datagrams
- Checksum is only over IP header, not over payload
- Typically 20 bytes of header
- IP options exist, most are blocked or unused
- ARP cache used to assist routing decisions (find MAC address of next hop)

Address Resolution Protocol

- Connects MAC and IP address of other hosts on the same wire (same IP network)
- Not routable
- Can notify other local hosts of our MAC and IP address (gratuitous ARPing, spam)
- Can look for stations using our IP address
- Results are cached for lookup per packet

Address Resolution Protocol

```
     Station: 00-80-C7-52-2C-9E ----> FF-FF-FF-FF-FF-FF
     Type: 0x0806 (ARP)
arp: ================== Address Resolution Protocol ====
     Hardware: Ethernet
     Protocol: 0x0800 (IP)
     Operation: ARP Request
     Hardware adress length: 6
     Protocol address length: 4
     Sender Hardware Address: 00-80-C7-52-2C-9E
     Sender Protocol Address: 129.123.1.71
     Target Hardware Address: 00-00-00-00-00-00
     Target Protocol Address: 129.123.1.49
```

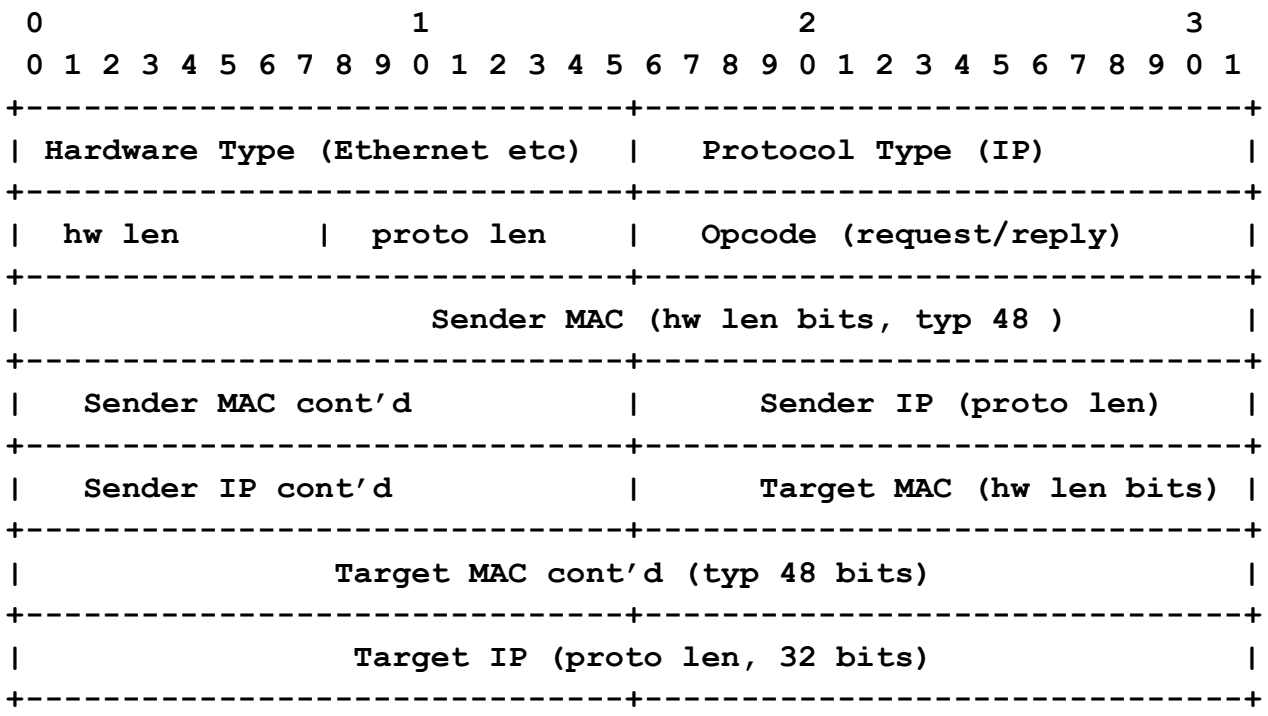
Ethertype 0806

Asks for MAC address of station 129.123.1.49

```
 0: FF FF FF FF FF FF 00 80 C7 52 2C 9E 08 06 00 01 |. . .
10: 08 00 06 04 00 01 00 80 C7 52 2C 9E 81 7B 01 47 |. . .
20: 00 00 00 00 00 00 81 7B 01 31                   |. . .
30:                                                 |&.W
```

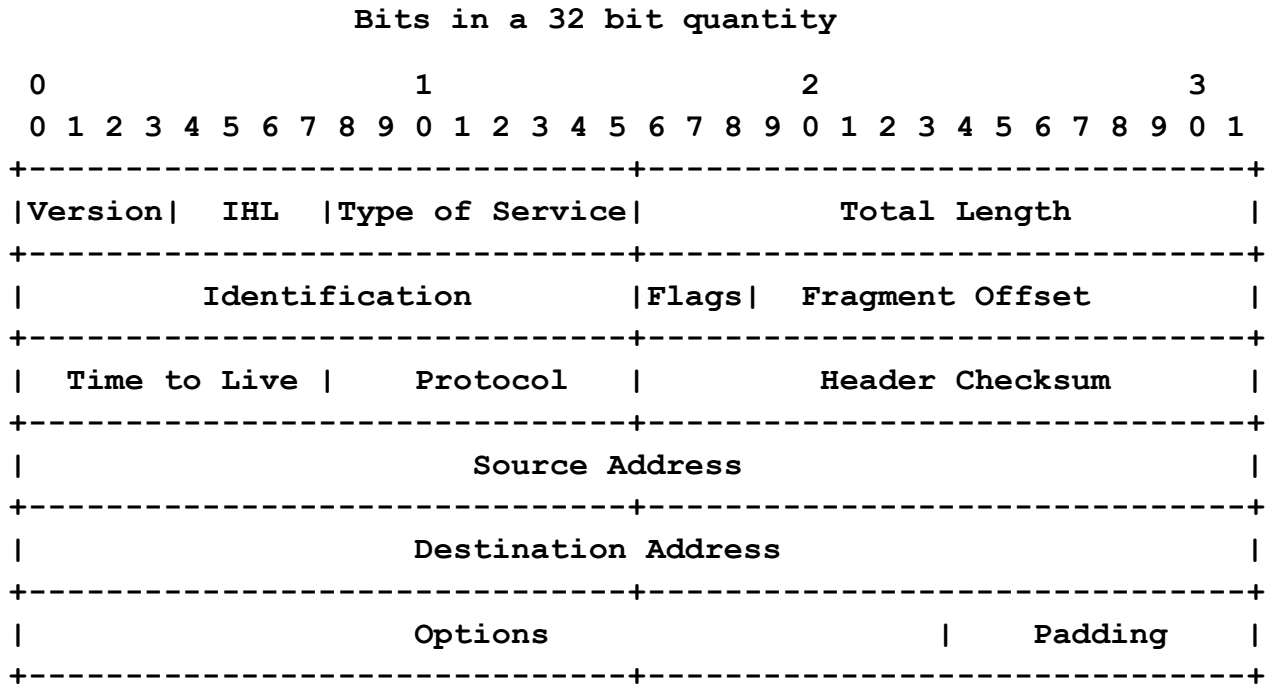
Novell

Address Resolution Protocol, not routable



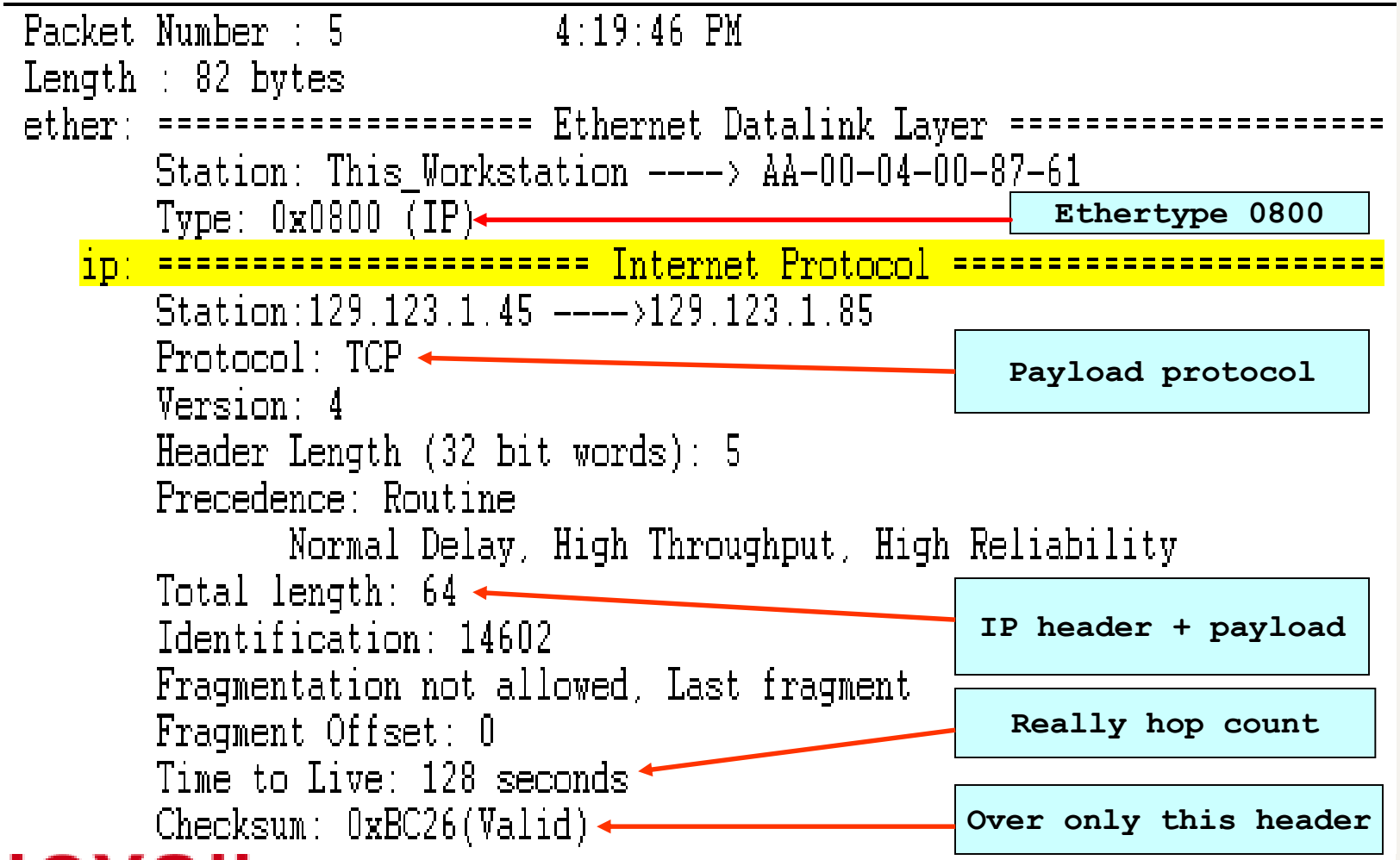
Novell

Internet Datagram Header



Novell

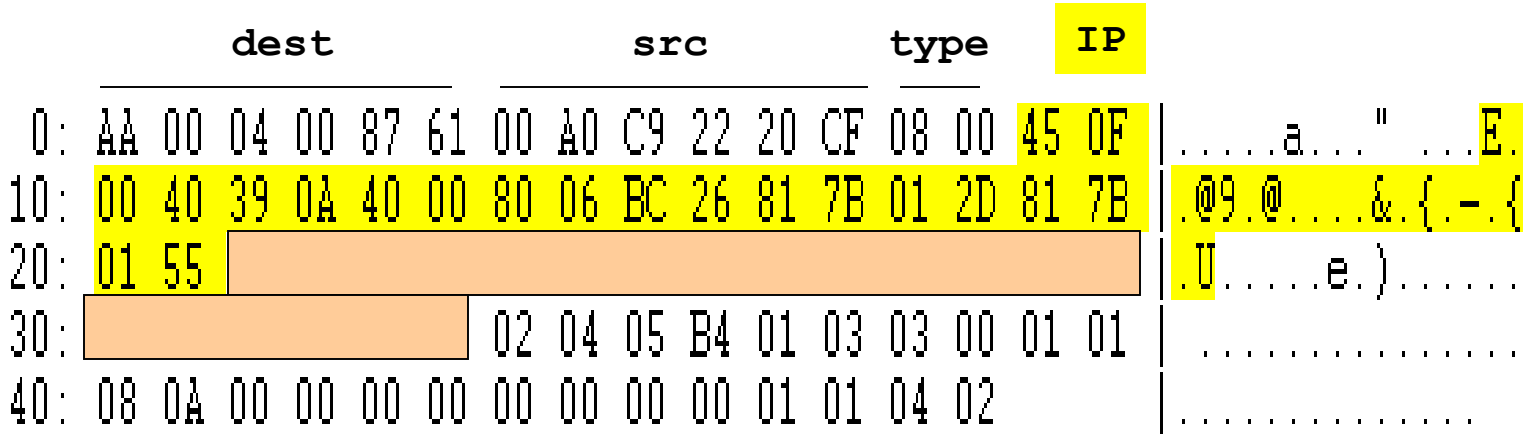
IP datagram details



Novell

IP datagram in TCP connection

Source	Destination	Layer	Summary	Error	Size	Interpack
This_Workstation	AA0004008761	tcp	Port:1288 —> TELNET SYN		82	59 µs
AA0004008761	This_Workstation	tcp	Port:TELNET —> 1288 ACK SYN		78	518 µs
This_Workstation	AA0004008761	tcp	Port:1288 —> TELNET ACK		70	124 µs
AA0004008761	This_Workstation	tcp	Port:TELNET —> 1288 ACK PUSH		94	1 ms
This_Workstation	AA0004008761	tcp	Port:1288 —> TELNET ACK PUSH		73	47 ms
AA0004008761	This_Workstation	tcp	Port:TELNET —> 1288 ACK PUSH		76	355 µs
This_Workstation	AA0004008761	tcp	Port:1288 —> TELNET ACK PUSH		73	3 ms
AA0004008761	This_Workstation	tcp	Port:TELNET —> 1288 ACK PUSH		73	20 ms
This_Workstation	AA0004008761	tcp	Port:1288 —> TELNET ACK PUSH		73	118 µs



Novell

IP addresses

- An IP V4 address is a 32-bit binary quantity
- It is not a numeric value, even though we humans write it in dotted decimal or hexadecmial forms
- An IP address represents both a network and a host field, and optionally a locally constructed “subnet” field
- IP fields are bit widths, not decimal values

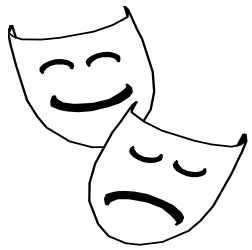
IP Address Classes

- Class A:0..b 0.host - 127.host
- Class B:10..b 128.x.host - 191.x.host
- Class C:110..b 192.x.x.host - 223.x.x.host
- Class D:111..b 224.multicast
- Class E:1111..b 240.reserved
- Classless Internet Domain Routing (CIDR) groups contiguous nets (typically Class C nets). Uses an explicit netmask in routers.

Simple IP routing (netmask)

- Every machine asks this routing question: “Is the destination IP on my IP network?”
- If yes we can send to it directly after obtaining its MAC address (ARP)
- If no we must use a gateway to relay for us. We get the gateway’s MAC address (ARP) and use the destination machine IP address. The router knows to send it onward (its job)

Netmask



- The way the decision is made uses a 32-bit netmask and it confuses almost everyone.
- IP address has both “network” and “host” identification in the same 32 bit quantity.
- Host means one attachment point to the net, with likely other attachments on the same net by this or other machines.

Network same/different calc

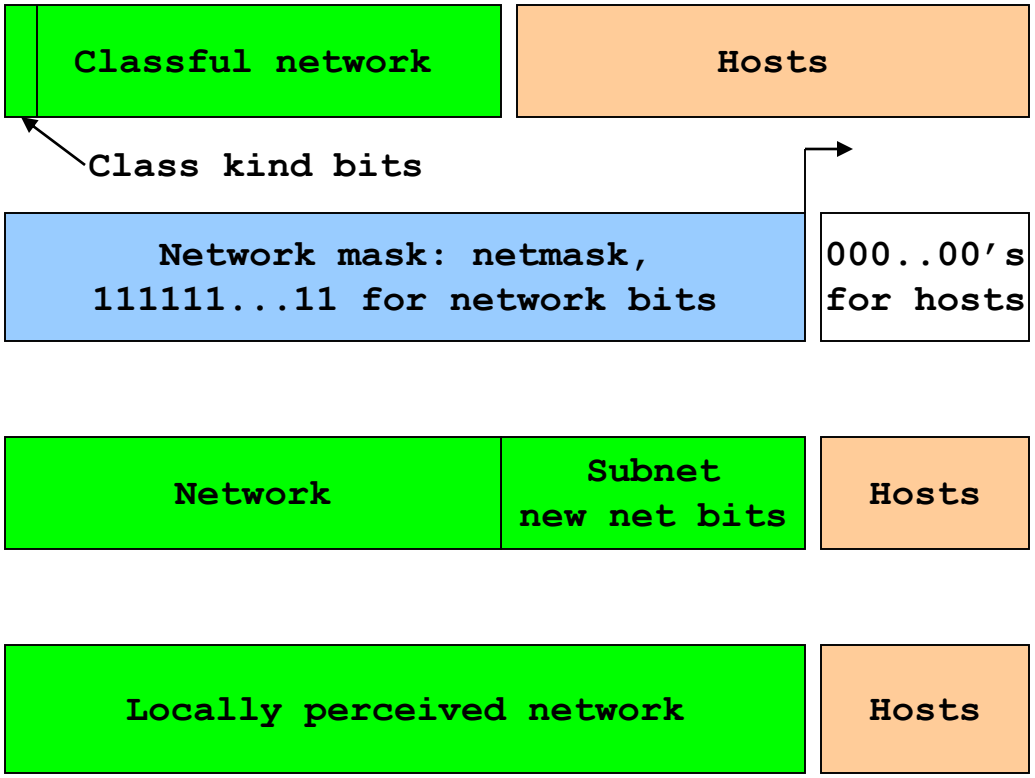
```
10000000 00111011 00100111 00000010  their_IP 128.59.39.2
10000001 01111001 00000001 00101011  my_IP 129.123.1.43
                                      XOR column at a time
00000001 01000010 00100110 00101001  -> differences
                                      (1 = different, 0 = same)
   AND column at a time with netmask to show only network diffs
11111111 11111111 11111111 00000000  netmask 255.255.255.0
        (networks)           (hosts)
 (mask is transparent)     (is opaque)
00000001 010000010 00100110 00000000 -> masked differences

Non-zero final result means the IP networks are not the
same and thus we must use a gateway/router to talk.
   If (((their_IP ^ my_IP) & netmask)) != 0)
        use_gateway();          /* long distance */
   else
        go_direct();            /* on same wire */
```

Novell

Subnetting

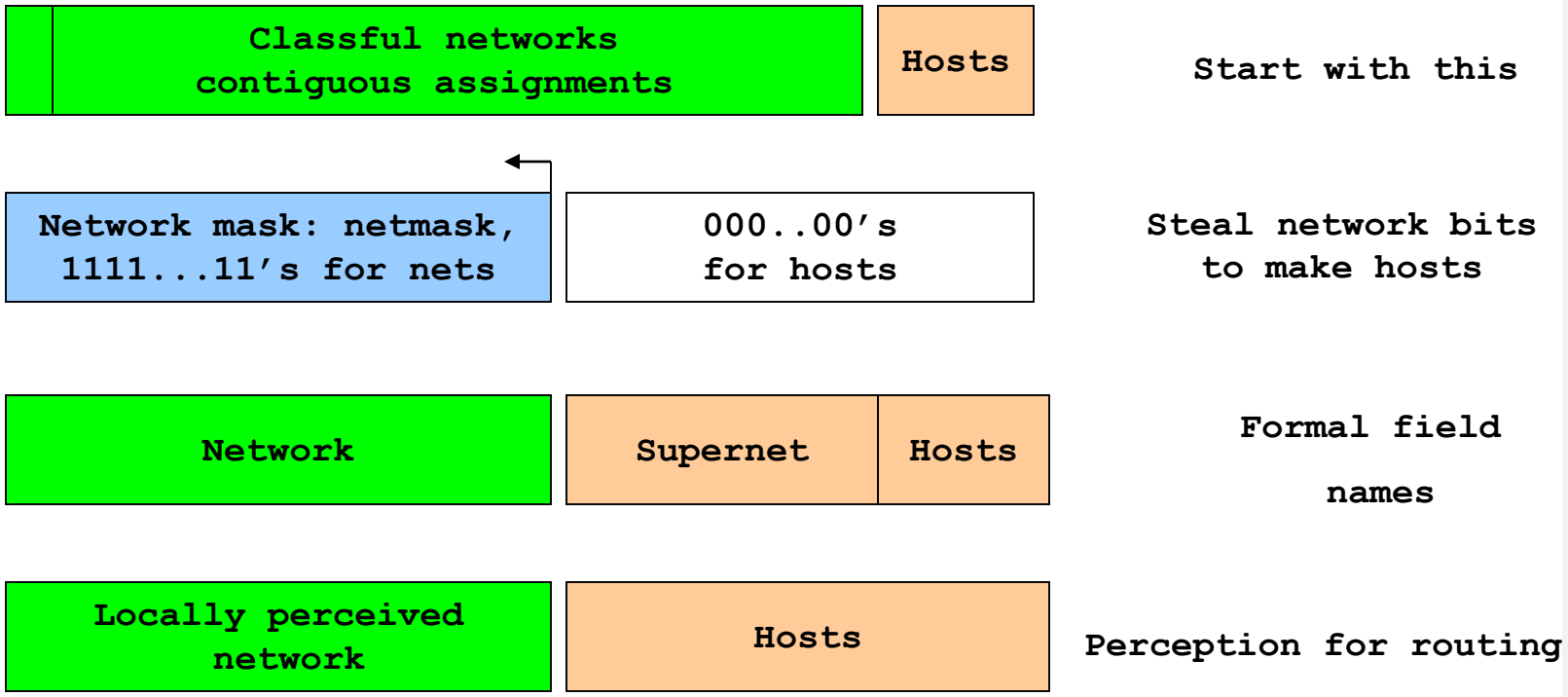
Single Class B example.



Novell

Supernetting

1,2,4,8... contiguous Class C addresses



Novell

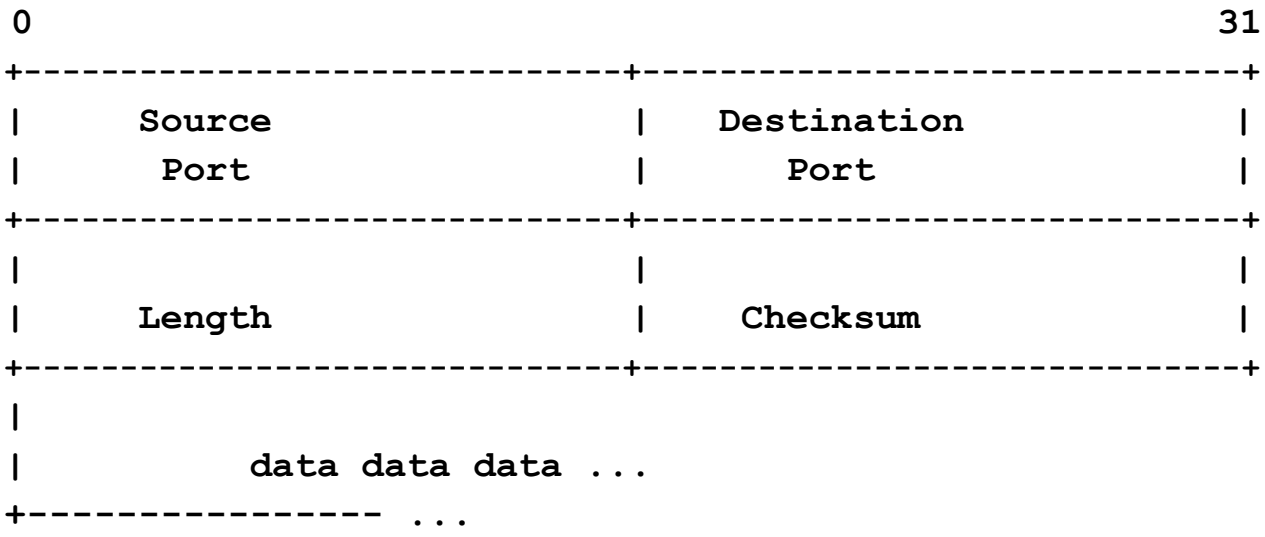
ICMP

- Internet Control Message Protocol
- IP to IP comms for network control
- Carried in IP datagram, so routable too
- Ping uses ICMP “Echo Request”
- “Source Quench” means slow down
- “Host unreachable”
- Many other detailed kinds
- Not accessible to normal user level programs

User Datagram Protocol (UDP)

- Does almost nothing
- Adds “ports” to support multiple apps at same time over UDP
- Adds optional checksum over entire UDP datagram
- Uses send and forget mode (datagrams)
- Each datagram is the entire message

User datagram Protocol



Length covers header (8 octets) and payload

Checksum covers header, payload, and unsent pseudo-header

Novell

UDP datagram

No.	Source	Destination	Layer	Summary
38	0080C7522C9E	FFFFFFFFFFFF	dhcp	Req DISCOVER-Locate Available S
39	00902725C3BC	0080C7522C9E	dhcp	Rply OFFER-Configuration Paramet
40	0080C7522C9E	FFFFFFFFFFFF	dhcp	Req REQUEST-Parameters from or

```
ether: ===================== Ethernet Datalink Layer ====================
       Station: 00-80-C7-52-2C-9E ----> FF-FF-FF-FF-FF-FF
       Type: 0x0800 (IP)
  ip:  ======================== Internet Protocol ========================
       Station:0.0.0.0 ---->255.255.255.255
       Protocol: UDP
       Version: 4
       Header Length (32 bit words): 5
       Precedence: Routine
              Normal Delay, Normal Throughput, Normal Reliability
       Total length: 328
       Identification: 43264
       Fragmentation allowed, Last fragment
       Fragment Offset: 0
       Time to Live: 128 seconds
       Checksum: 0x90A5(Valid)
 udp:  ===================== User Datagram Protocol ======================
       Source Port: BOOTPC
       Destination Port: BOOTPS
       Length = 308
       Checksum: 0x14D2(Valid)
 dhcp: ============== Dynamic Host Configuration Protocol ==============
       Message op code: 1 (BOOTREQUEST)
       Hardware Address Type: 1 : Ethernet (10Mb)
```

```
  0: FF FF FF FF FF FF 00 80 C7 52 2C 9E 08 00 45 00 |.........R,...E.
 10: 01 48 A9 00 00 00 80 11 90 A5 00 00 00 00 FF FF |.H..............
 20: FF FF 00 44 00 43 01 34 14 D2 01 01 06 00 62 3B |...D.C.4.....b;
```

Protocol ident

Length IP
Length UDP

Ports

Novell

UDP Transmission Limits

- No ACKs, no feedback, no timer, fragile
- No network throttle: blast & pray
- Must use small buffers (4-8KB) to avoid saturating routers and over running slow receivers
- NFS v2 has the major flow control problems, NFS v3 uses TCP to eliminate them

Transmission Control Protocol

- The major protocol of the suite
- Validated robust service
- Checksum covers header and payload
- Continuous session (data delivered in sequence without gaps or duplication)
- Has timers to discover missing packets and to quickly replace them (dynamic)

TCP, cont'd

- Has "ports" to support multiple applications using TCP at the same time
- Transmission unit is named a "segment"
- IP can send a segment in one or more pieces (fragments if more than one)
- Max Segment Size (MSS) negotiated at connection startup, can be 64KB, typ. 536B(576-40) to 1460B(1500-40)

TCP, cont'd

- Each session is full duplex: an independent data channel for each direction
- No concept of message boundaries: data is a stream of octets sent however and whenever TCP wishes
- Typically 4-32KB buffers for transmit and receive, can be much larger
- Receive buffer capacity (window) in pkts

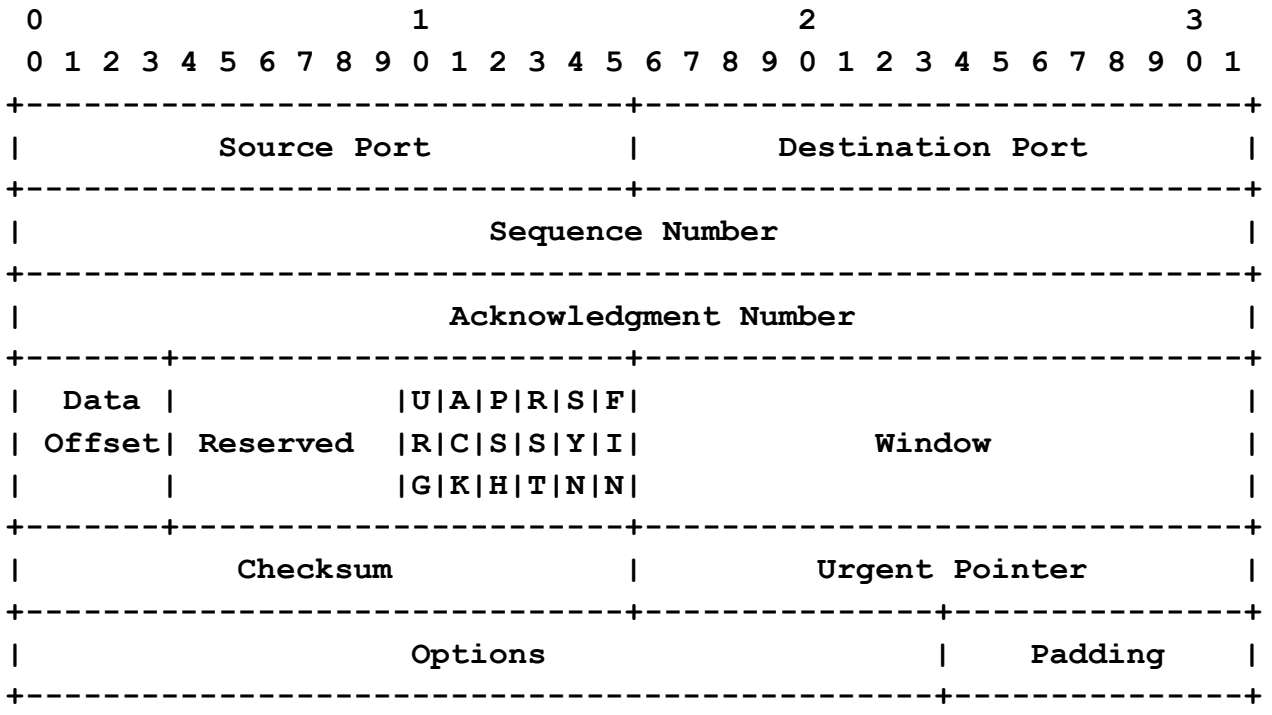
TCP, cont'd

- Supports a number of options
- Typical header size is 20 bytes
- Every header carries both sequence number (of data being sent, if any) and acknowledgment number (of last data octet +1 received in order)
- Numbering is by byte in stream
- IP header provides TCP length value

TCP, cont'd

- Because sessions span individual segments/packets there is state kept for each session
- Session startup and shutdown requires 3-4 packets, called a "three way handshake"
- Poor clients can leave sessions half open (SYN attack style) or half closed

Transmission Control Protocol



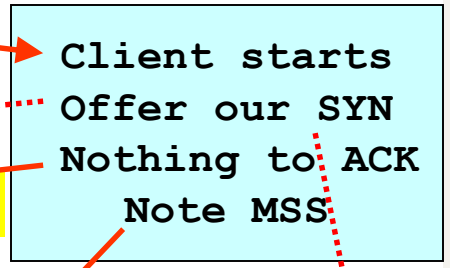
Checksum covers header, payload, unsent pseudo-header

IP supplies length to TCP

Novell

TCP initial segment

```
tcp: ================= Transmission Control Protocol =
     Source Port: 1288
     Destination Port: TELNET
     Sequence Number: 6620201
     Acknowledgement Number: 0
     Data Offset (32-bit words): 11
     Window: 8192
     Control Bits: Synchronize Sequence Numbers (SYN)
     Checksum: 0x06DC(Valid)
     Urgent Pointer: 0
     Option:MAXIMUM SEGMENT SIZE
            Option Length: 4
            Maximum Segment Size : 1460
```

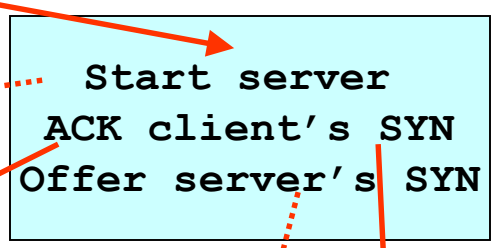


1500(Eth)-20(IP)-20(TCP)=1460

Novell

TCP startup exchanges

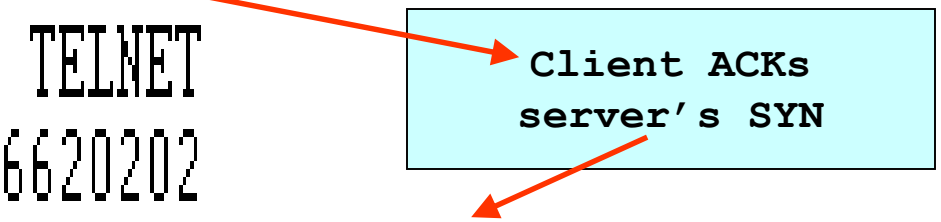
```
tcp: ================ Transmission Control Protocol ===
     Source Port: TELNET
     Destination Port: 1288
     Sequence Number: 212039446
     Acknowledgement Number: 6620202
     Data Offset (32-bit words): 10
     Window: 6144
     Control Bits: Acknowledgement Field is Valid (ACK)
           Synchronize Sequence Numbers (SYN)
```



Novell

TCP startup exchanges

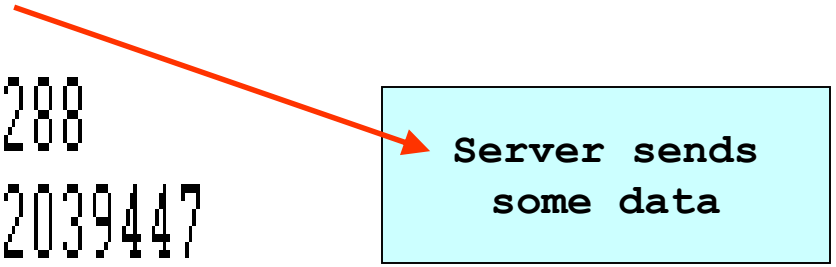
```
tcp: ================ Transmission Control Protocol ==
     Source Port: 1288
     Destination Port: TELNET
     Sequence Number: 6620202
     Acknowledgement Number: 212039447
     Data Offset (32-bit words): 8
     Window: 8760
     Control Bits: Acknowledgement Field is Valid (ACK)
```



Novell

TCP startup exchanges

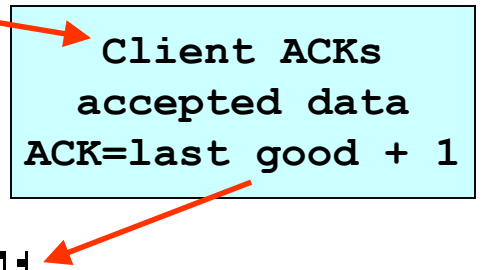
```
tcp: ================ Transmission Control Protocol ==
     Source Port: TELNET
     Destination Port: 1288
     Sequence Number: 212039447
     Acknowledgement Number: 6620202
     Data Offset (32-bit words): 8
     Window: 6144
     Control Bits: Acknowledgement Field is Valid (ACK)
```



Novell

TCP startup exchanges

```
tcp: ================ Transmission Control Protocol ==
     Source Port: 1288
     Destination Port: TELNET
     Sequence Number: 6620202
     Acknowledgement Number: 212039471
     Data Offset (32-bit words): 8
     Window: 8736
     Control Bits: Acknowledgement Field is Valid (ACK)
```



Novell

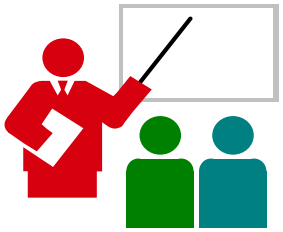
TCP retransmissions

- Retransmission after loss of a packet obeys a truncated exponential backoff schedule:
  - try once at timeout delay
  - double delay for next attempt, double on each following attempt
  - truncate to one minute per try
- Total retry time can be many minutes

TCP what to do while doing nothing

- When there is nothing to say on the wire then nothing is said on the wire
- No “hello” or link integrity packets
- Routers and links can go up and down (including boom-boom stuff) and the end stations do not care (datagrams)
- Some stacks may employ keep-alive probes to test for logging out

Protocol basics



- Items necessary for robust protocols
  - Checksums for data integrity
  - Checksums on both data and ACKs
    - IP: covers only IP header
    - UDP: optional, covers UDP header and data
    - TCP: covers TCP header and data
    - Simple linear addition (1's complement of 1's complement sum)

Protocol basics

- ACKs to confirm delivery and provide flow control, must have sequence numbers to avoid confusion about what is sent and ACKed.
  - IP: none. Pure connectionless datagram
  - UDP: none. Pure connectionless datagram
  - TCP: full, connection oriented. Rules say all TCP data must be ACKed sooner or later, even if old, repeated, or far future data. Soon means < 0.5 sec and that is often 200ms in wide practice.

Protocol basics

- Sequence numbers to distinguish old, new, duplicate data
  - IP: none. IP ident number is different for each datagram, used to reassemble fragments
  - UDP: none, each datagram is the entire message
  - TCP: full, 32-bit, identifies starting octet in this segment, starting point is random and set in SYN segment. Packets are not otherwise numbered.

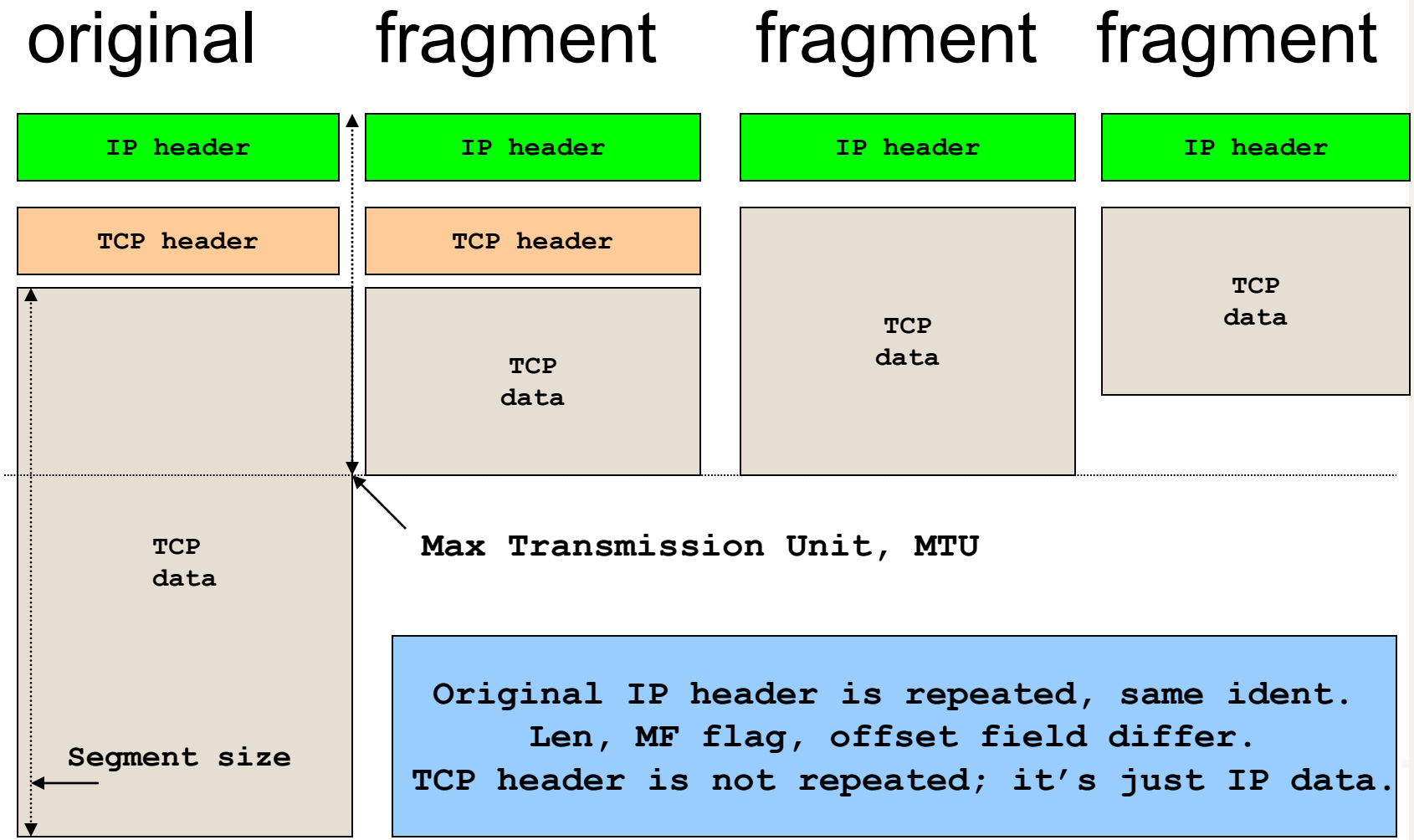
Protocol basics

- Timers to break deadlocks from lost packets
  - IP: none, no feedback
  - UDP: none, no feedback
  - TCP: full. Measure round trip delay to stop waiting for lost packets. ACKs may be delayed to group many into one, keep-alive probes, etc. Granularity is often tens to 200 milliseconds, which is very coarse.
  - TCP uses arriving ACKs to clock out new data, operates at full network speed

Protocol basics

- Flow control
  - IP: none except a few ICMP “source quench” pkts
  - UDP: never heard of the topic. Manual throttling required. Poor through congested networks.
  - TCP: full featured
    - Dynamic estimation of network capacity (Van Jacobson’s work). Congestion avoidance adapts to changing network conditions.
    - Each packet announces receiver buffer space available: window size
    - Arriving ACKs can announce resource space

IP Fragmentation



Original IP header is repeated, same ident.
Len, MF flag, offset field differ.
TCP header is not repeated; it's just IP data.

Novell

IP fragmentation

- 64KB max IP datagram (16-bit length field)
- If wire capacity is smaller then either generate smaller IP datagrams (MTU Path Discovery) or fragment this datagram
- Only receiver reassembles fragments, not done by routers

IP fragmentation

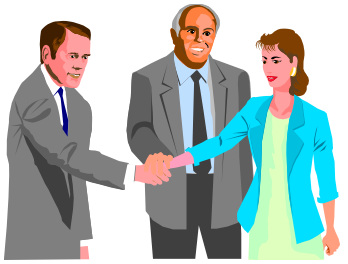
- Fragmentation is expensive in time and memory; avoid by generating smaller datagrams
- One lost component causes all parts to be lost
- Fragmentation is on 8 byte boundaries
- Routers can fragment if NDF bit is clear
- IPV6 requires transmitter to fragment, not routers (not clever)

TCP data streams

- SYN/FIN punctuate a stream of data
  - SYN bit FIN bit

    data data data data data
- No record boundaries
- Bytes are put into packets as TCP sees fit and are sent when TCP and IP wish to do so
- SYN segment has starting sequence number, and both SYN and FIN bits are ACKed as pseudo data bytes

Three Way Handshake



- --->SYN (my seq number)
- <--- ACK (for their seq num +1)
  <--- SYN (my seq number)
- ---> ACK (for their seq num + 1)
- Each end opens its own stream to the other and uses its own starting sequence number
- Random start confuses wire snoopers

TCP session startup

ARP for NS, DNS request & reply, ARP for host, TCP SYN

PUSH means have sent all data from application

Source	Destination	Layer	Summary	Error	Size	Interpacket Time
08002074096A	This_Workstation	arp	Reply 129.123.1.2=08002074096A		64	0 µs
This_Workstation	08002074096A	dns	Std Req A cc.usu.edu		74	74 µs
08002074096A	This_Workstation	dns	Std Rply cc.usu.edu A 129.123.1.85		344	3 ms
AA0004008761	This_Workstation	arp	Reply 129.123.1.85=AA0004008761		64	30 ms
This_Workstation	AA0004008761	tcp	Port:1288 –> TELNET SYN			µs
AA0004008761	This_Workstation	tcp	Port:TELNET –> 1288 ACK SYN			µs
This_Workstation	AA0004008761	tcp	Port:1288 –> TELNET ACK		70	124 µs
AA0004008761	This_Workstation	tcp	Port:TELNET –> 1288 ACK PUSH		94	1 ms
This_Workstation	AA0004008761	tcp	Port:1288 –> TELNET ACK PUSH		73	47 ms
AA0004008761	This_Workstation	tcp	Port:TELNET –> 1288 ACK PUSH		76	355 µs

3 way handshake

Novell

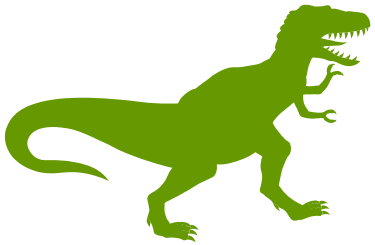
Three Way Handshake

- --->FIN (my seq number), “no more data”
- <--- ACK (for their seq num +1)
- <--- FIN (my seq number) (after data)
- ---> ACK (for their seq num + 1)
- Each end closes its stream independently
- FIN means no more data from here, but will listen for more arriving data
- A missing ACK to a FIN can cause holdup

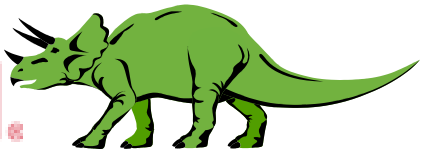
Three Way Handshake

- SYN and FIN three way handshakes are tinygrams and take time to create/decode and route across the network.
- Web clients get faster service by using a keep-alive connection: making a request/reply channel from a single persistent connection and putting one request after another onto it.

TCP Heuristic Park



- Heuristics can be defined as “Gee, it seemed like a good idea at the time.”
- We look at two sets: for flow control with congestion avoidance, and for speedy yet plump packets on the wire.
- These try to make the system work better, faster, smoother.



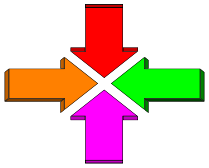
TCP Flow Control

- Van Jacobson: packets are lost from net congestion, rarely from bit-rot or routing confusion
- Test network capacity by sending packets
- ACK says packet has left the net (space on net is now available)
- ACK grants permission to send a replacement and often another datagram
- Net can drop a datagram from overload and further growth should be slow

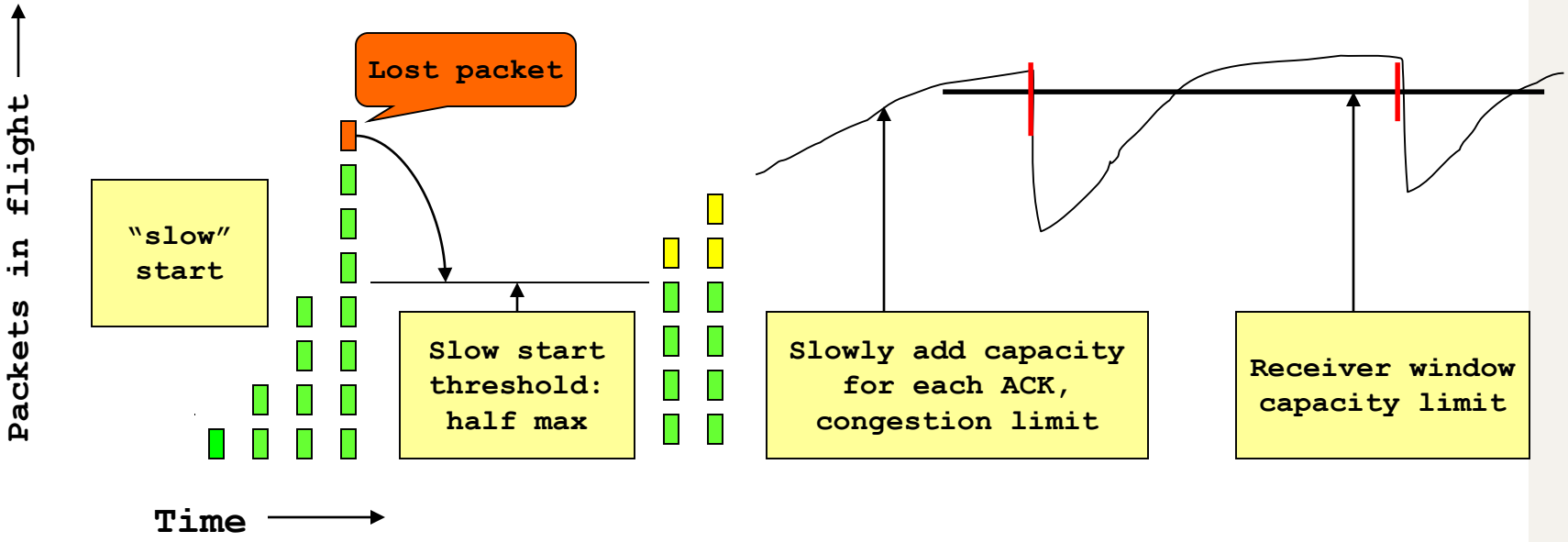
TCP Flow Control

- After a packet loss drop back to slow rate of testing network capacity
- The drop back is very quick to maintain network stability under impulsive loads
- Normal operation fills a “congestion window’s” worth of transmission credits or fills the receiver’s window
- Each arriving ACK yields a new send opportunity. Sends become clocked by ACKs

Congestion Avoidance



- Van Jacobson: ramp up, find network capacity, drop back, slowly increase
- Capacity: min(network, receiver window)

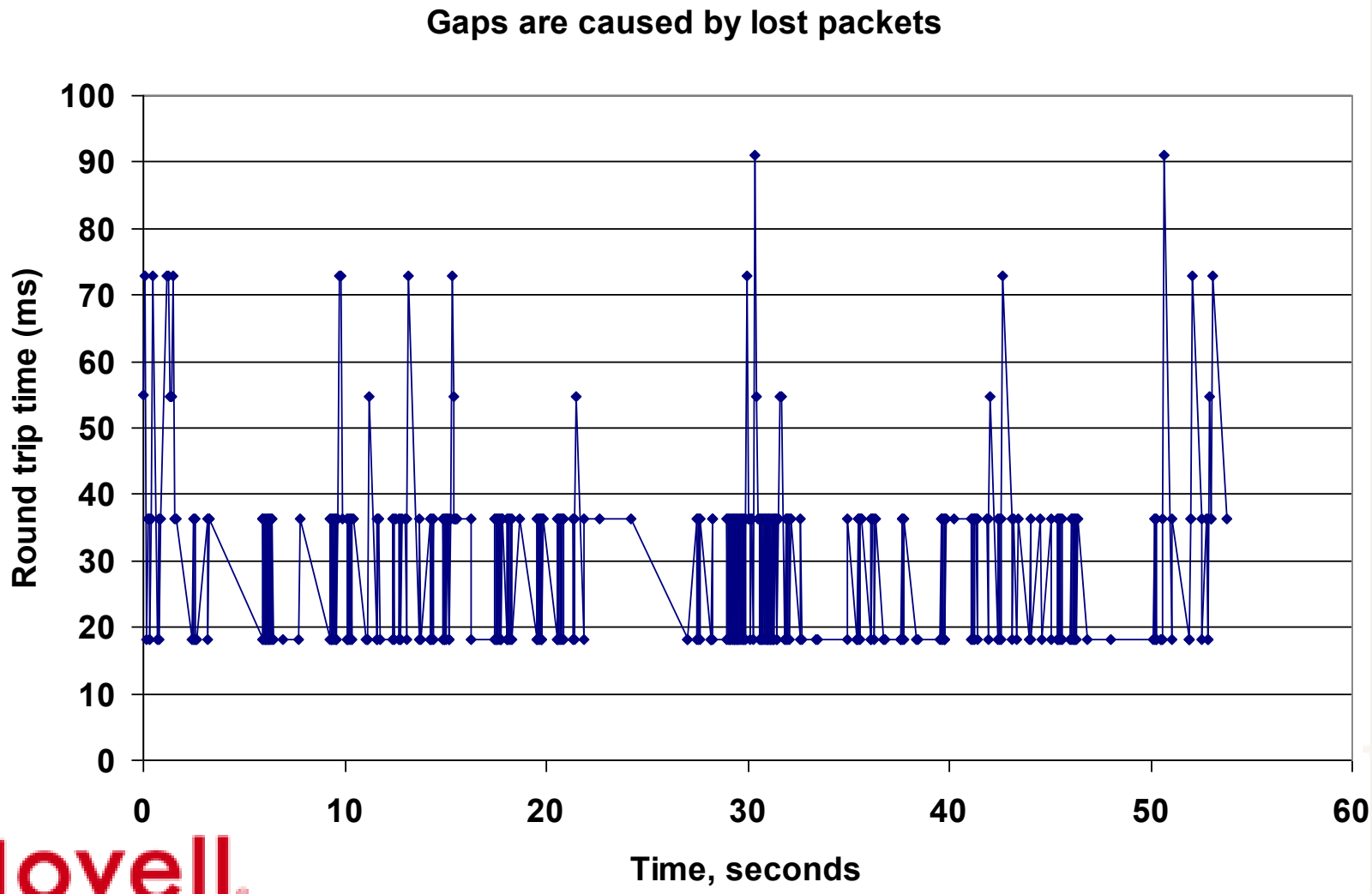


Novell

Congestion Avoidance

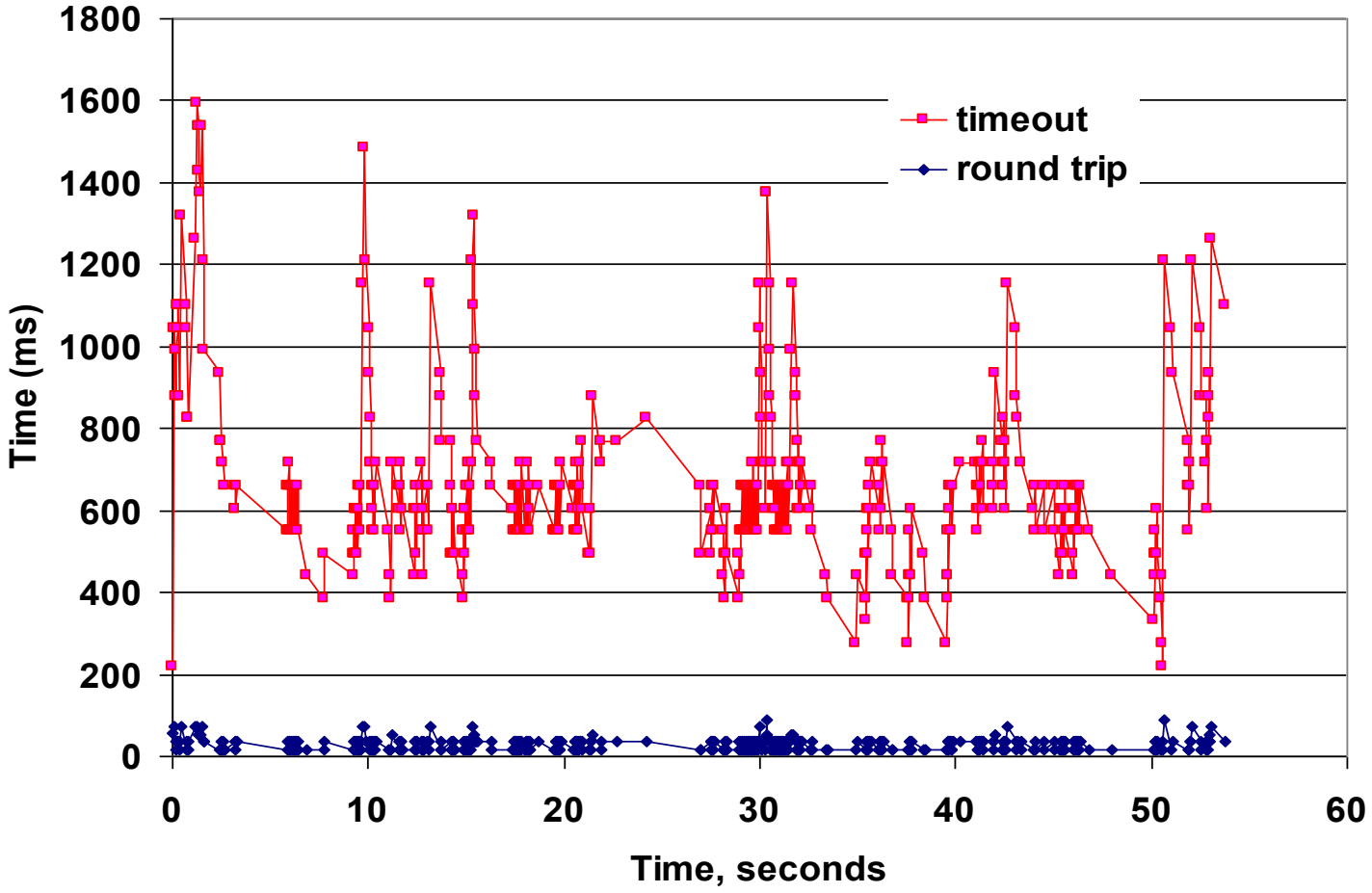
- Measure round trip time (ACK arrival)
- Use rtt to estimate time to wait for missing ACKs (and hence when to retransmit)
- Allow for chaotic style network traffic (variance in rtt) to avoid too many repeated transmissions
- Timeout varies with network conditions

TCP, NYC to Utah



Novell

TCP, NYC to Utah



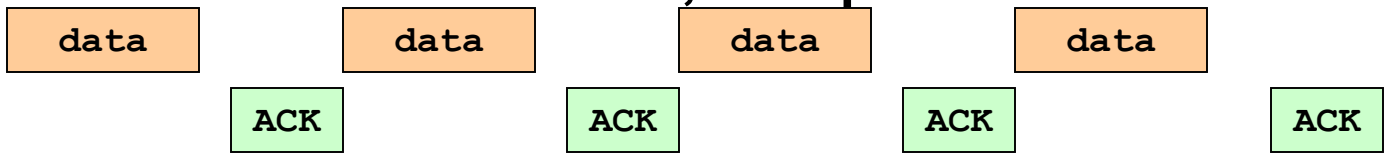
Novell

Statistical queueing results

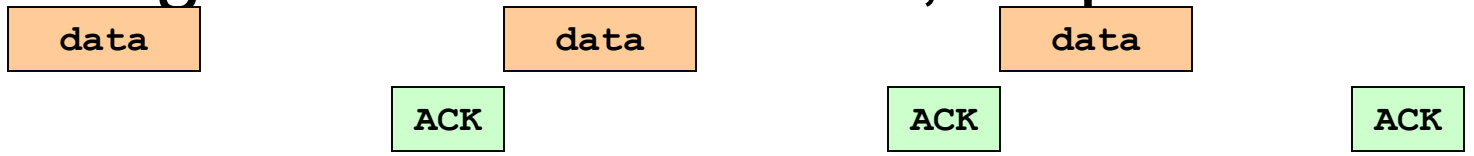
- a = average arrival rate (say packets/sec)
- s = average service rate (say packets/sec)
- Number of items waiting in queue
  = s/(s-a) = 1/(1 - a/s)
- Time to exit system = 1/(s - a)
- Queue length & waiting time go infinite as a nears s
- Traffic queues in routers; delay, overflows.

Path length effect on throughput

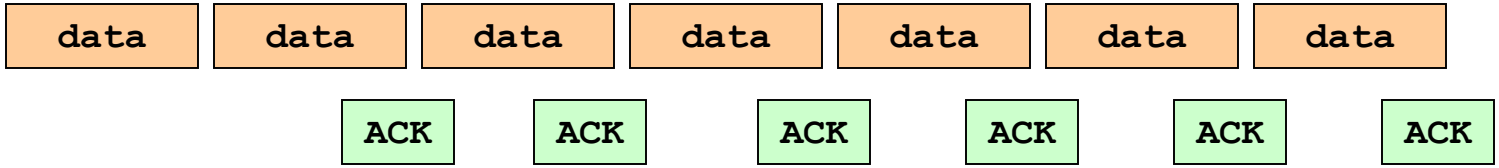
- Direct connection, stop & wait:



- Bridged/switched/routed, stop & wait:



- Any connection, streaming:



Time ⟶

Novell

TCP more heuristics

- Delayed ACKs save sending extra tinygrams (recall: must ACK “sooner or later”, often later). Receiver hopes more data will come soon.
- Nagle condition says delay sending small packets, hoping more app data will arrive to make full packets. Hold small packets until all previous data have been ACKed.

TCP heuristics cont'd

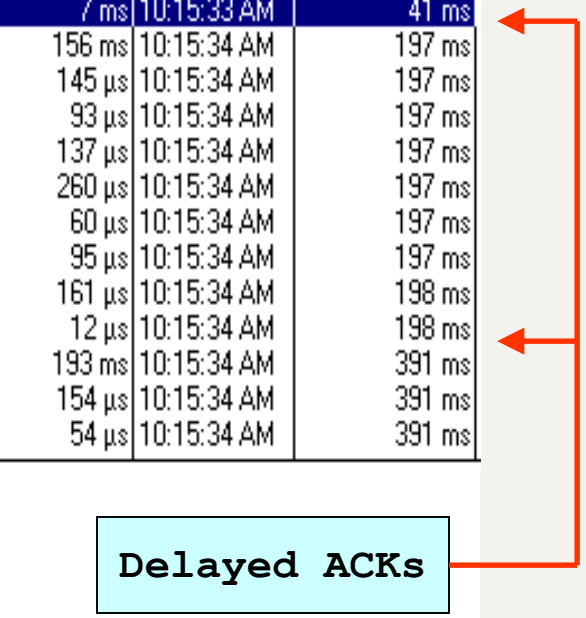
- Nagle condition holding back plus delayed ACKs creates a deadlock situation where each end waits on the other.
- Both ends "guess" more app data will arrive shortly, but often there isn't any
- Deadlock is broken by delayed ACK timer firing, often 200ms later
- Deadlock in request/response systems often leads to five exchanges/sec max.

TCP heuristics finished off

- Recent work by the author replaces Nagle mode with a new transmission policy which sends small TCP packets only when the application says it has no more data available.
- No dependence on ACKs and variable network delays, no guesses, no deadlock is possible, full packets, goes fast.
- Draft-doupnik-tcpimpl-nagle-mode-00.txt in the IETF material at http://www.ietf.org.

Nagle+delayed ACK deadlock

No.	Source	Destination	Layer	Summary	Error	Size	Interpacket Time	Absolute Time	Relative Time
1	netlab6	Netlab1	arp	Reply 129.123.1.109=009027190777		64	0 µs	10:15:33 AM	0 µs
2	Netlab1	netlab6	tcp	Port:39968 ---> 6601 SYN		64	63 µs	10:15:33 AM	63 µs
3	netlab6	Netlab1	tcp	Port:6601 ---> 39968 ACK SYN		64	228 µs	10:15:33 AM	290 µs
4	Netlab1	netlab6	tcp	Port:39968 ---> 6601 ACK		64	106 µs	10:15:33 AM	396 µs
5	Netlab1	netlab6	tcp	Port:39968 ---> 6601 ACK PUSH		165	177 µs	10:15:33 AM	573 µs
6	netlab6	Netlab1	tcp	Port:6601 ---> 39968 ACK PUSH		137	33 ms	10:15:33 AM	33 ms
7	Netlab1	netlab6	tcp	Port:39968 ---> 6601 ACK		64	59 µs	10:15:33 AM	33 ms
8	netlab6	Netlab1	tcp	Port:6601 ---> 39968 ACK PUSH		82	110 µs	10:15:33 AM	33 ms
9	Netlab1	netlab6	tcp	Port:39968 ---> 6601 ACK		64	60 µs	10:15:33 AM	33 ms
10	netlab6	Netlab1	tcp	Port:6601 ---> 39968 ACK		1,518	7 ms	10:15:33 AM	41 ms
11	Netlab1	netlab6	tcp	Port:39968 ---> 6601 ACK		64	156 ms	10:15:34 AM	197 ms
12	netlab6	Netlab1	tcp	Port:6601 ---> 39968 ACK PUSH		828	145 µs	10:15:34 AM	197 ms
13	Netlab1	netlab6	tcp	Port:39968 ---> 6601 ACK		64	93 µs	10:15:34 AM	197 ms
14	Netlab1	netlab6	tcp	Port:39968 ---> 6601 ACK PUSH		165	137 µs	10:15:34 AM	197 ms
15	netlab6	Netlab1	tcp	Port:6601 ---> 39968 ACK PUSH		137	260 µs	10:15:34 AM	197 ms
16	Netlab1	netlab6	tcp	Port:39968 ---> 6601 ACK		64	60 µs	10:15:34 AM	197 ms
17	netlab6	Netlab1	tcp	Port:6601 ---> 39968 ACK PUSH		82	95 µs	10:15:34 AM	197 ms
18	netlab6	Netlab1	tcp	Port:6601 ---> 39968 ACK		1,518	161 µs	10:15:34 AM	198 ms
19	Netlab1	netlab6	tcp	Port:39968 ---> 6601 ACK		64	12 µs	10:15:34 AM	198 ms
20	Netlab1	netlab6	tcp	Port:39968 ---> 6601 ACK		64	193 ms	10:15:34 AM	391 ms
21	netlab6	Netlab1	tcp	Port:6601 ---> 39968 ACK PUSH		828	154 µs	10:15:34 AM	391 ms
22	Netlab1	netlab6	tcp	Port:39968 ---> 6601 ACK		64	54 µs	10:15:34 AM	391 ms



Novell

New transmission policy

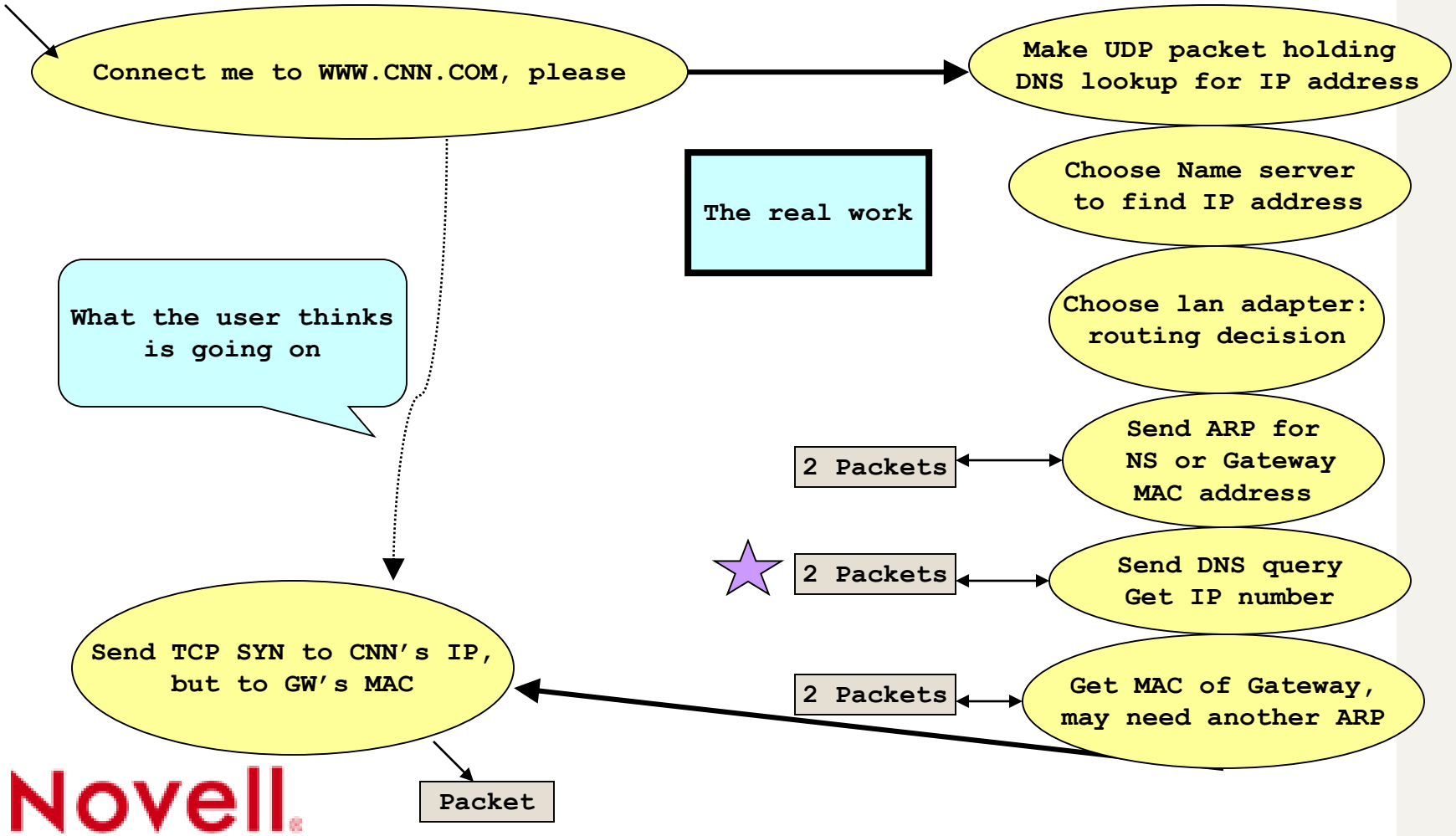
Capture Buffer

No.	Source	Destination	Layer	Summary	Error	Size	Interpacket Time	Absolute Time	Relative Time
1	Netlab1	netlab6	tcp	Port:39974 ---> 6601 SYN		64	0 µs	10:20:59 AM	0 µs
2	netlab6	Netlab1	tcp	Port:6601 ---> 39974 ACK SYN		64	231 µs	10:20:59 AM	231 µs
3	Netlab1	netlab6	tcp	Port:39974 ---> 6601 ACK		64	139 µs	10:20:59 AM	370 µs
4	Netlab1	netlab6	tcp	Port:39974 ---> 6601 ACK PUSH		165	178 µs	10:20:59 AM	548 µs
5	netlab6	Netlab1	tcp	Port:6601 ---> 39974 ACK PUSH		137	2 ms	10:20:59 AM	2 ms
6	netlab6	Netlab1	tcp	Port:6601 ---> 39974 ACK PUSH		82	12 µs	10:20:59 AM	2 ms
7	Netlab1	netlab6	tcp	Port:39974 ---> 6601 ACK		64	68 µs	10:20:59 AM	2 ms
8	Netlab1	netlab6	tcp	Port:39974 ---> 6601 ACK		64	12 µs	10:20:59 AM	2 ms
9	netlab6	Netlab1	tcp	Port:6601 ---> 39974 ACK		1,518	378 µs	10:20:59 AM	3 ms
10	netlab6	Netlab1	tcp	Port:6601 ---> 39974 ACK PUSH		828	45 µs	10:20:59 AM	3 ms
11	Netlab1	netlab6	tcp	Port:39974 ---> 6601 ACK		64	76 µs	10:20:59 AM	3 ms
12	Netlab1	netlab6	tcp	Port:39974 ---> 6601 ACK PUSH		165	71 µs	10:20:59 AM	3 ms
13	netlab6	Netlab1	tcp	Port:6601 ---> 39974 ACK PUSH		137	243 µs	10:20:59 AM	3 ms
14	netlab6	Netlab1	tcp	Port:6601 ---> 39974 ACK PUSH		82	12 µs	10:20:59 AM	3 ms
15	Netlab1	netlab6	tcp	Port:39974 ---> 6601 ACK		64	50 µs	10:20:59 AM	3 ms
16	Netlab1	netlab6	tcp	Port:39974 ---> 6601 ACK		64	11 µs	10:20:59 AM	3 ms
17	netlab6	Netlab1	tcp	Port:6601 ---> 39974 ACK		1,518	213 µs	10:20:59 AM	3 ms
18	netlab6	Netlab1	tcp	Port:6601 ---> 39974 ACK PUSH		646	13 µs	10:20:59 AM	3 ms
19	netlab6	Netlab1	tcp	Port:6601 ---> 39974 ACK PUSH		240	40 µs	10:20:59 AM	3 ms
20	Netlab1	netlab6	tcp	Port:39974 ---> 6601 ACK		64	54 µs	10:20:59 AM	4 ms
21	Netlab1	netlab6	tcp	Port:39974 ---> 6601 ACK		64	12 µs	10:20:59 AM	4 ms
22	Netlab1	netlab6	tcp	Port:39974 ---> 6601 ACK PUSH		165	122 µs	10:20:59 AM	4 ms

Novell

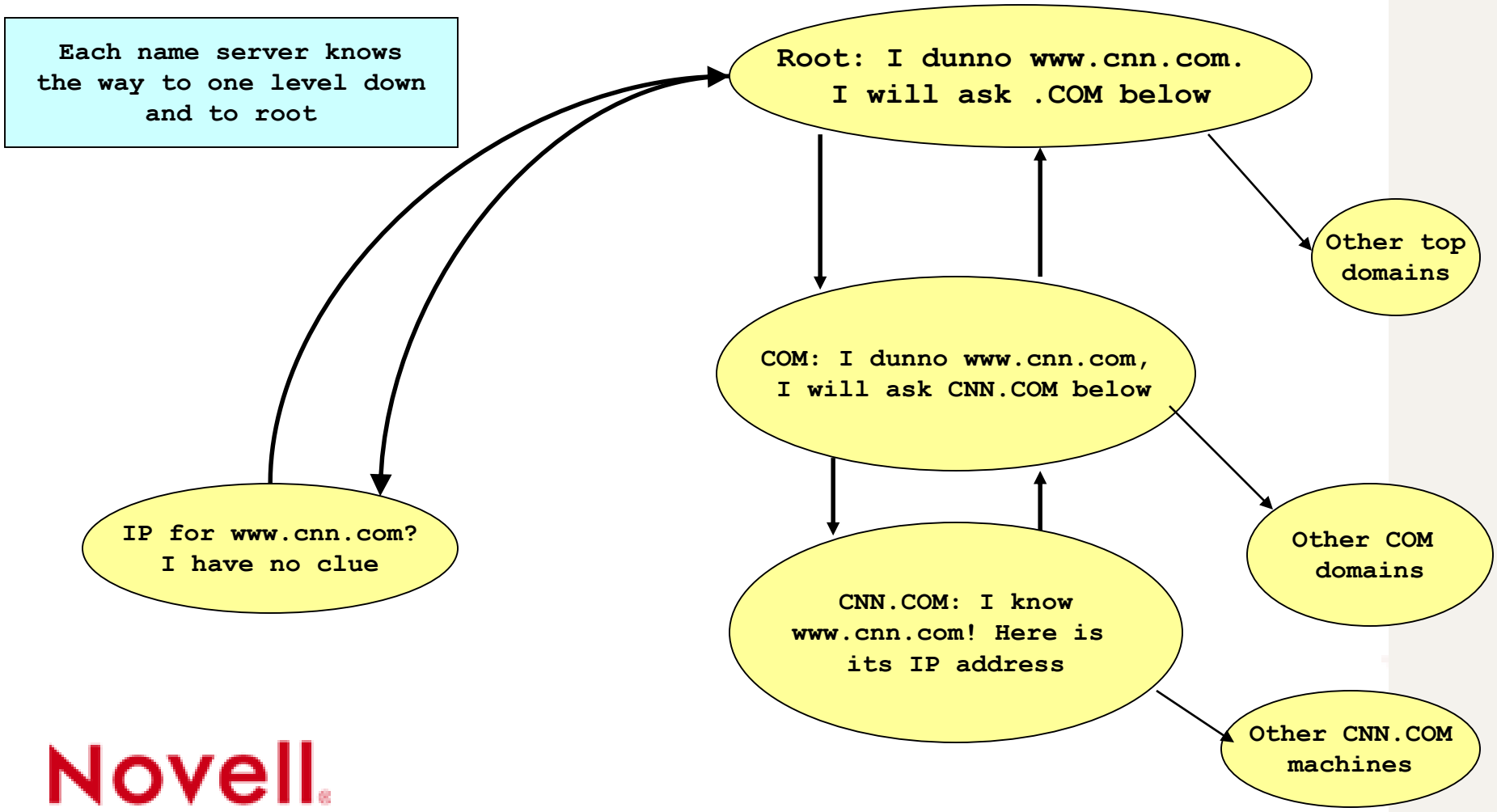
No waiting on Delayed ACKs

Why connection startup is slow



Novell

DNS name resolution



Novell

DNS name resolution

- DNS servers cache/remember answers to recent queries
- Caching-only DNS server asks a local friend (forwarding) or root (otherwise) for answers it does not have cached. This is a good item to keep on a wire.
- Reference DNS server is BIND, Berkeley Internet Name Daemon, see www.isc.org

Ports and Five-tuples

- One client with two Telnet sessions to same remote host:
  - protocol (TCP) same for both sessions
  - src IP same for both sessions
  - dest IP same for both sessions
  - dest port (23) same for both sessions
  - src port different for each session
- Thus the five-tuple distinguishes each session

TCP and UDP Ports

- Port numbers are unique to each protocol, so port 20 for UDP is unrelated to port 20 for TCP
- A service must be registered for each port, else no place to deliver the data (and a packet will be rejected in that case)
- Traceroute bounces off a randomly chosen port number to receive an ICMP “Port Unreachable” message

Ports, cont'd

- Some well known ports are
  - 13 Daytime (for Rdate), TCP & UDP
  - 21 FTP server, TCP
  - 23 Telnet server, TCP
  - 25 SMTP mail, TCP
  - 53 DNS server,TCP & UDP
  - 80 HTTP web server, TCP
  - 123 NTP server (Network Time Protocol), UDP
- TCP ports are independent of UDP ports

All this number stuff, made easy

- MAC address selects adapter on wire
- MAC Type field selects protocol stack
- IP num selects attachment point on net
- IP Protocol field selects higher protocol
- Port selects which application over this protocol
- This is just a bunch of direction signs

Novell