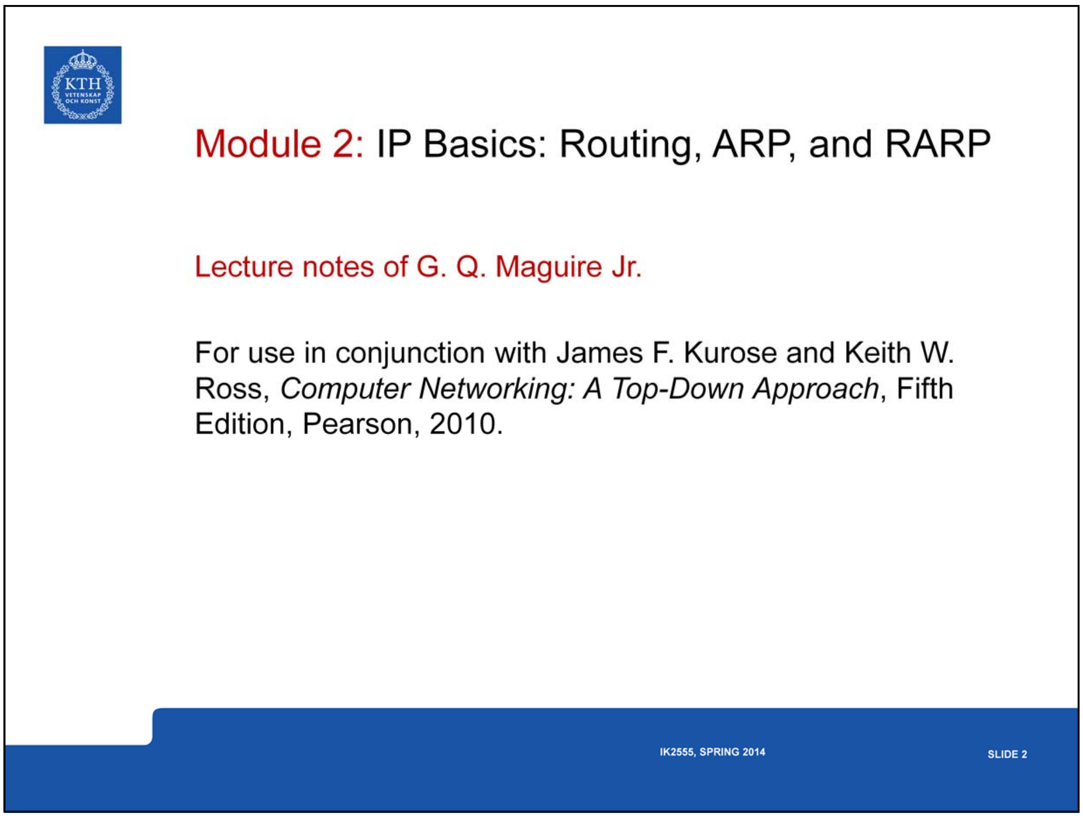
IK1550 & IK1552 Internetworking/Internetteknik

prof. Gerald Q. Maguire Jr. http://web.ict.kth.se/~maguire

School of Information and Communication Technology (ICT), KTH Royal Institute of Technology

IK1550/IK1552 Spring 2014, Period 4 2014.03.20



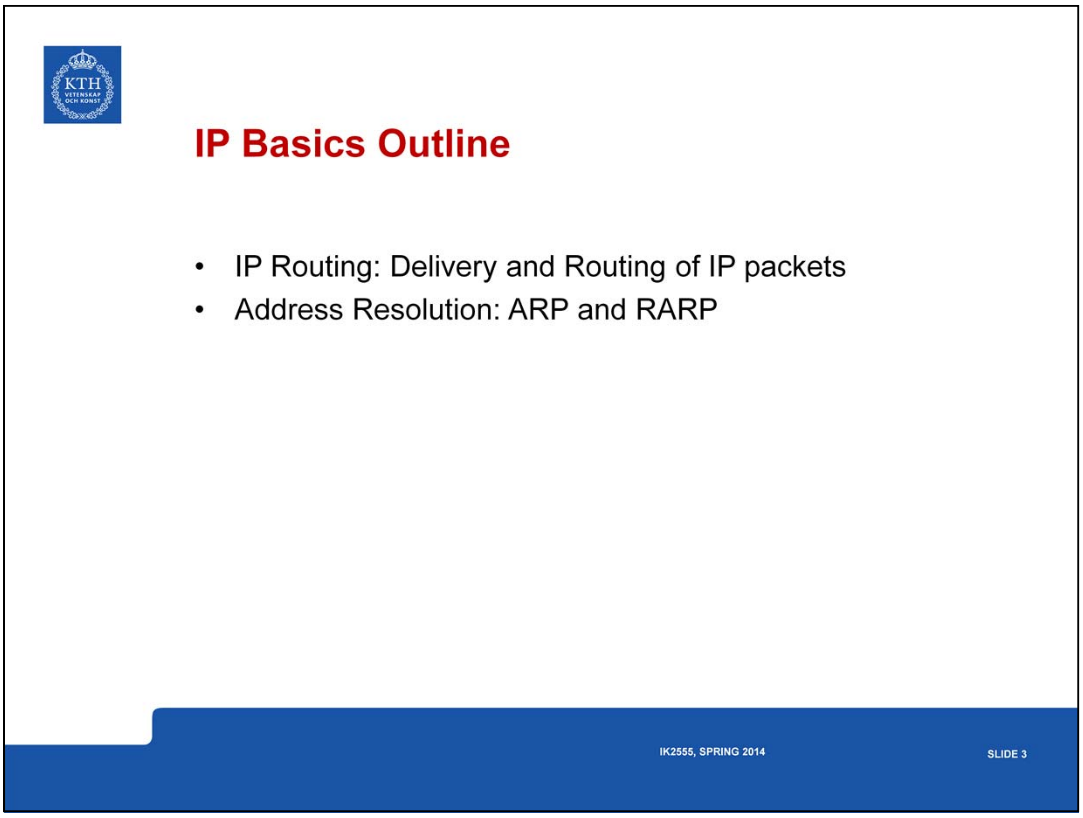
Module 2: IP Basics: Routing, ARP, and RARP

Lecture notes of G. Q. Maguire Jr.

For use in conjunction with James F. Kurose and Keith W. Ross, *Computer Networking: A Top-Down Approach*, Fifth Edition, Pearson, 2010.

IK2555, SPRING 2014

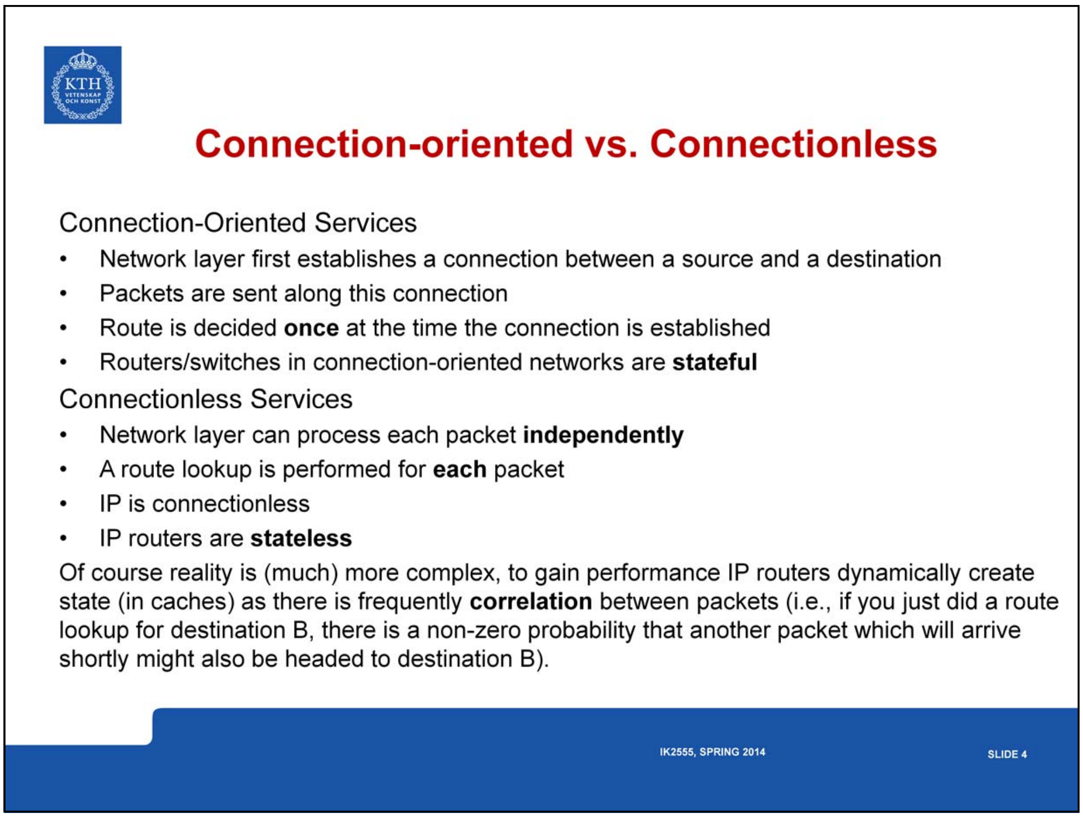
SLIDE 2



IP Basics Outline

- IP Routing: Delivery and Routing of IP packets
- Address Resolution: ARP and RARP

IK2555, SPRING 2014 SLIDE 3



Connection-oriented vs. Connectionless

Connection-Oriented Services

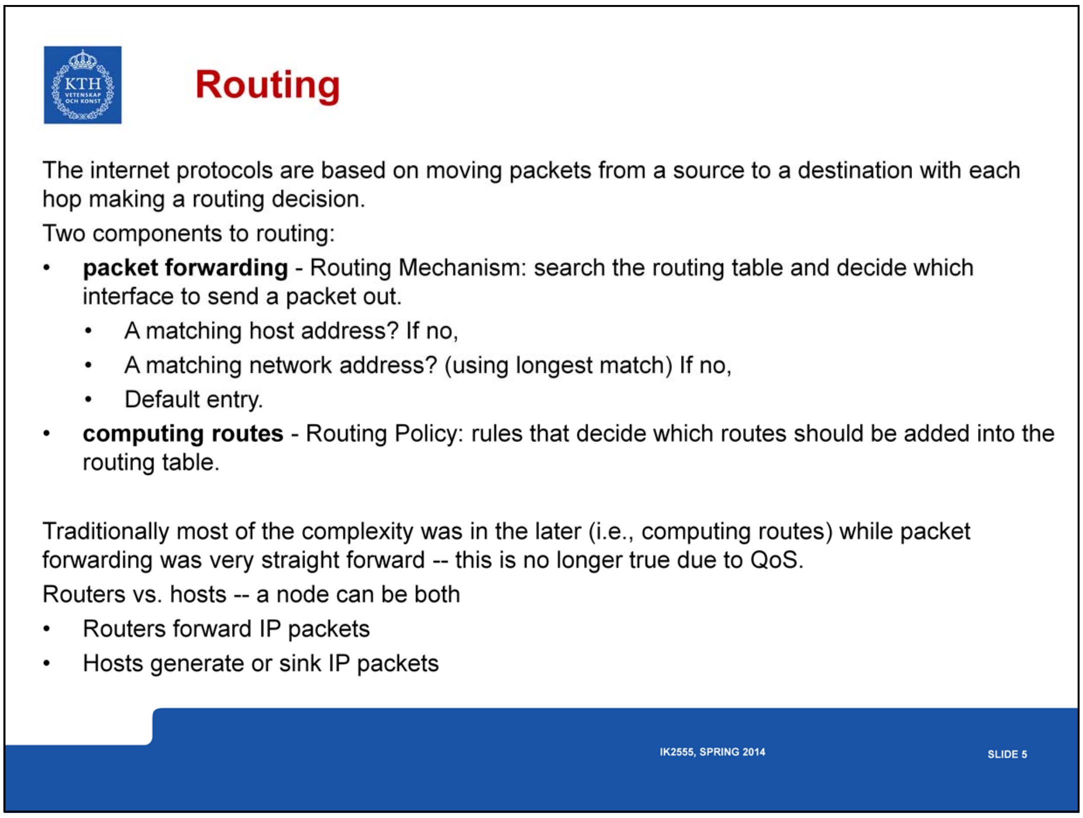
- Network layer first establishes a connection between a source and a destination
- Packets are sent along this connection
- Route is decided **once** at the time the connection is established
- Routers/switches in connection-oriented networks are **stateful**

Connectionless Services

- Network layer can process each packet **independently**
- A route lookup is performed for **each** packet
- IP is connectionless
- IP routers are **stateless**

Of course reality is (much) more complex, to gain performance IP routers dynamically create state (in caches) as there is frequently **correlation** between packets (i.e., if you just did a route lookup for destination B, there is a non-zero probability that another packet which will arrive shortly might also be headed to destination B).

IK2555, SPRING 2014 SLIDE 4



Routing

The internet protocols are based on moving packets from a source to a destination with each hop making a routing decision.

Two components to routing:

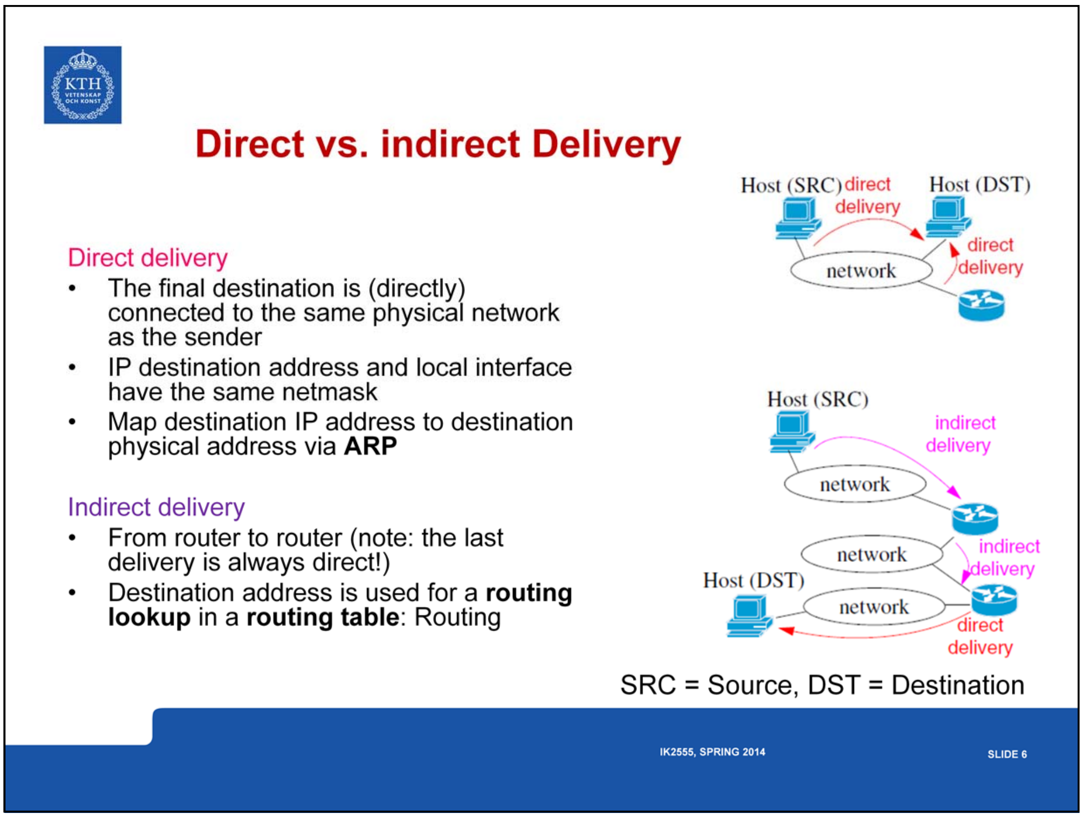
- **packet forwarding** - Routing Mechanism: search the routing table and decide which interface to send a packet out.
  - A matching host address? If no,
  - A matching network address? (using longest match) If no,
  - Default entry.
- **computing routes** - Routing Policy: rules that decide which routes should be added into the routing table.

Traditionally most of the complexity was in the later (i.e., computing routes) while packet forwarding was very straight forward -- this is no longer true due to QoS.

Routers vs. hosts -- a node can be both

- Routers forward IP packets
- Hosts generate or sink IP packets

IK2555, SPRING 2014 SLIDE 5



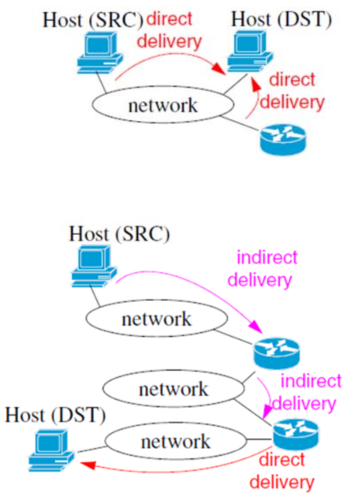
Direct vs. indirect Delivery

Direct delivery

- The final destination is (directly) connected to the same physical network as the sender
- IP destination address and local interface have the same netmask
- Map destination IP address to destination physical address via **ARP**

Indirect delivery

- From router to router (note: the last delivery is always direct!)
- Destination address is used for a **routing lookup** in a **routing table**: Routing

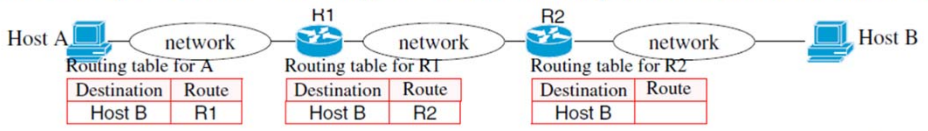


SRC = Source, DST = Destination

IK2555, SPRING 2014 SLIDE 6

Forwarding

- Next-Hop method - routing table holds only the address of the next hop



Routing table for A

Destination	Route
Host B	R1

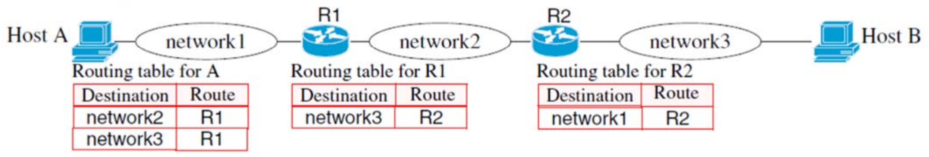
Routing table for R1

Destination	Route
Host B	R2

Routing table for R2

Destination	Route
Host B	

- Network-specific method - routing table entries are for networks



Routing table for A

Destination	Route
network2	R1
network3	R1

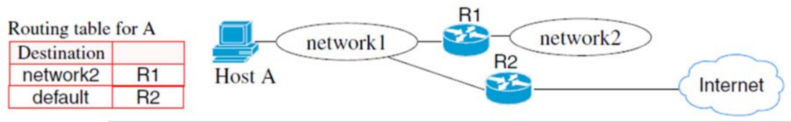
Routing table for R1

Destination	Route
network3	R2

Routing table for R2

Destination	Route
network1	R2

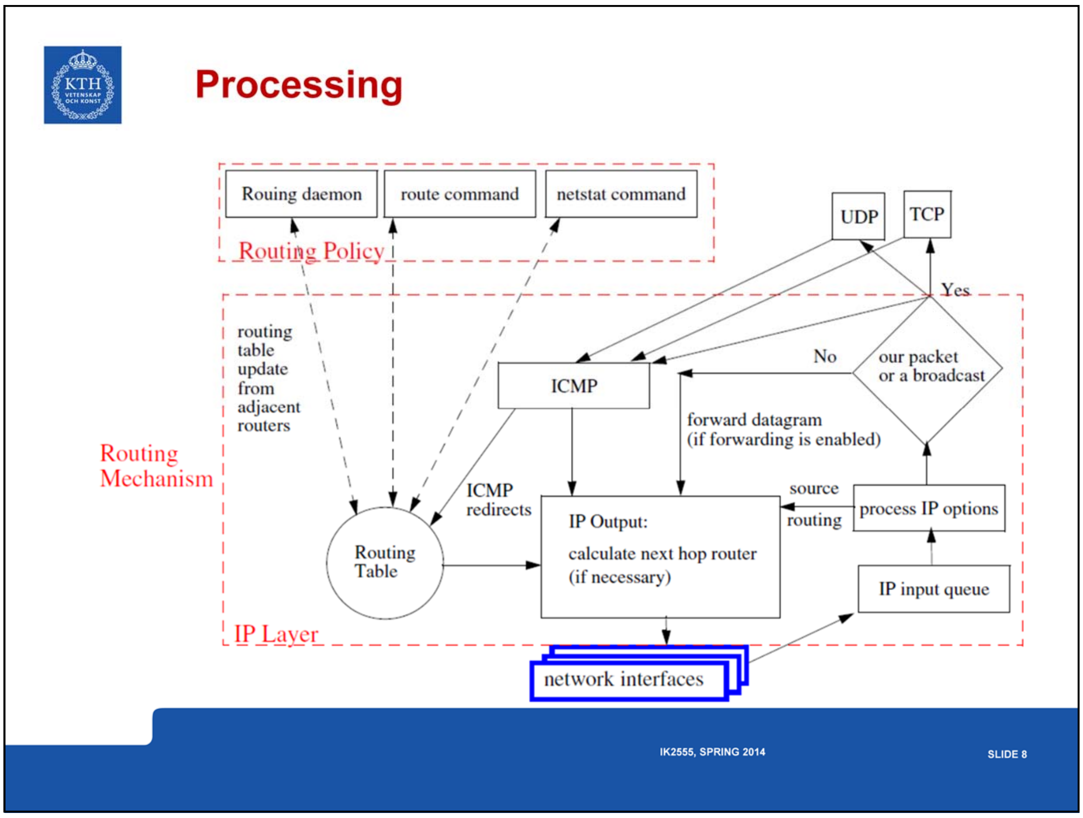
- Host-specific method - per host routes
- Default method - specifies a default route (normally network address 0.0.0.0)



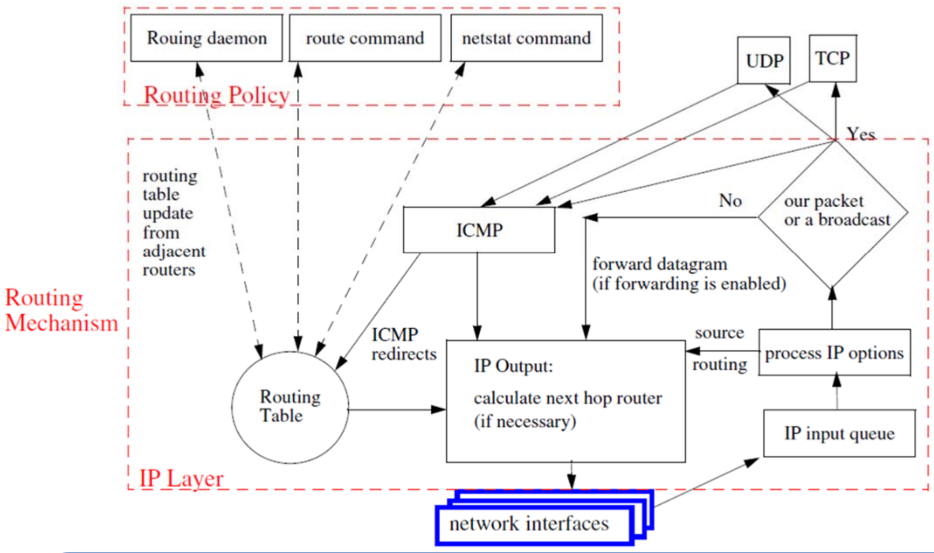
Routing table for A

Destination	
network2	R1
default	R2

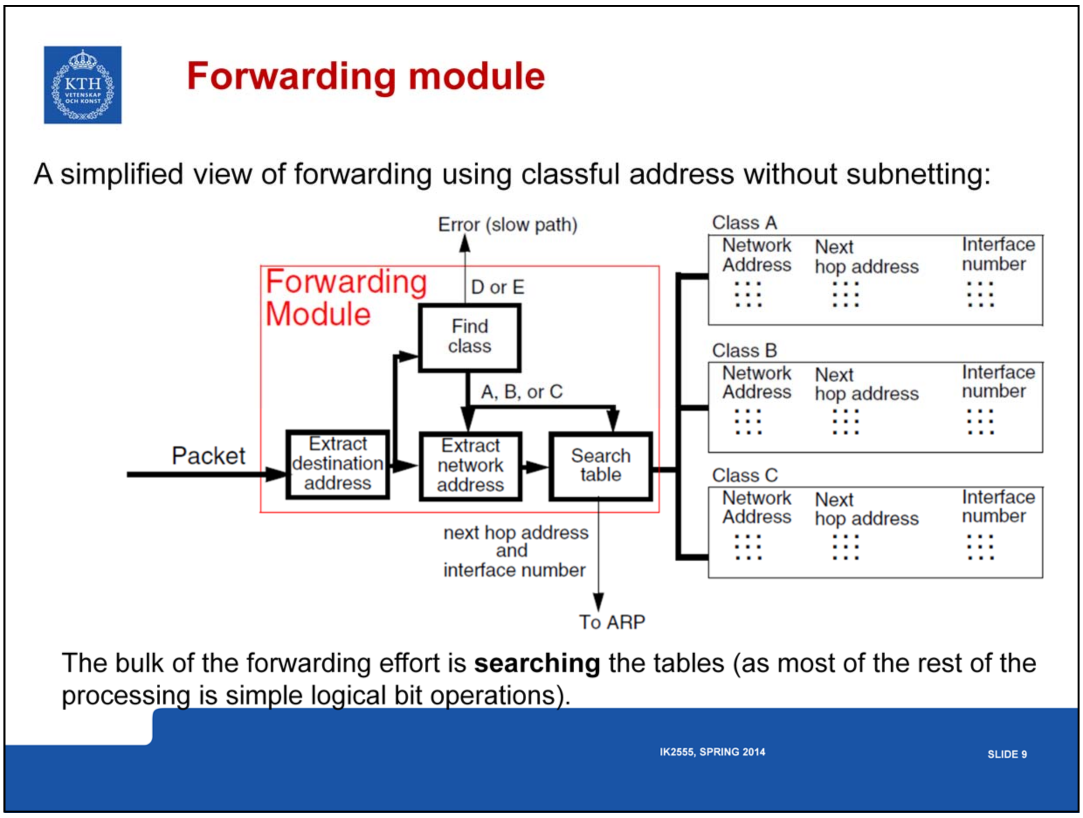
IK2555, SPRING 2014 SLIDE 7



Processing

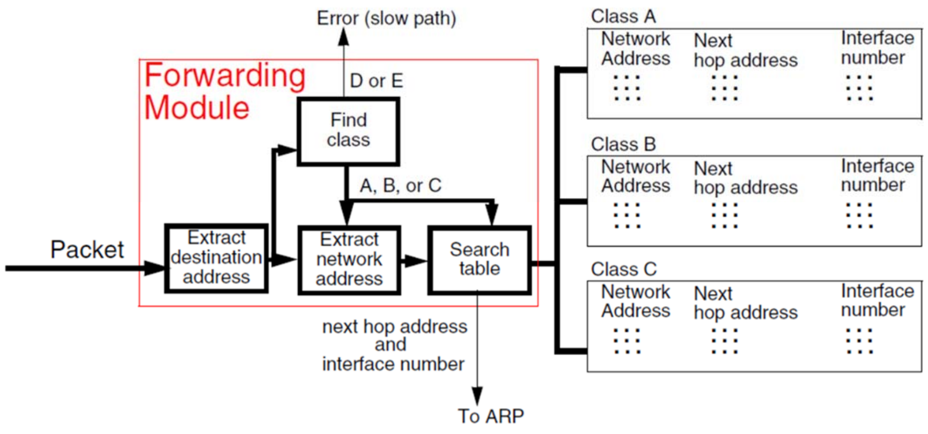


IK2555, SPRING 2014 SLIDE 8

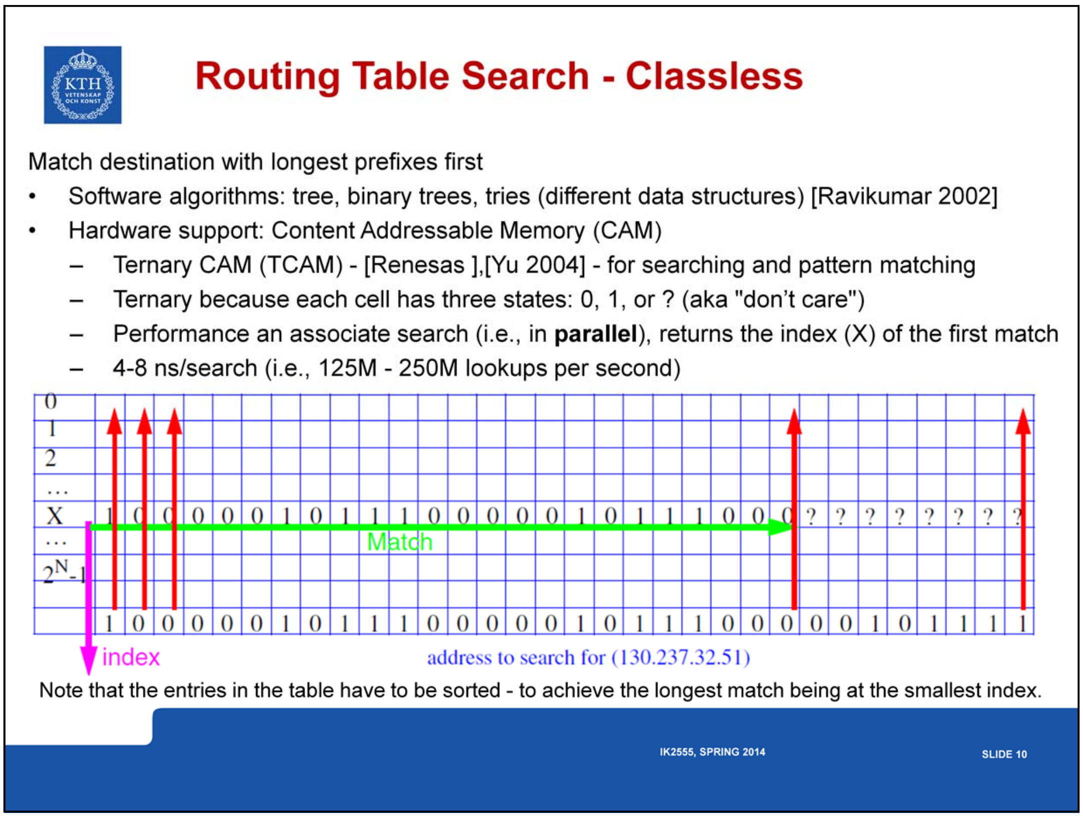


Forwarding module

A simplified view of forwarding using classful address without subnetting:



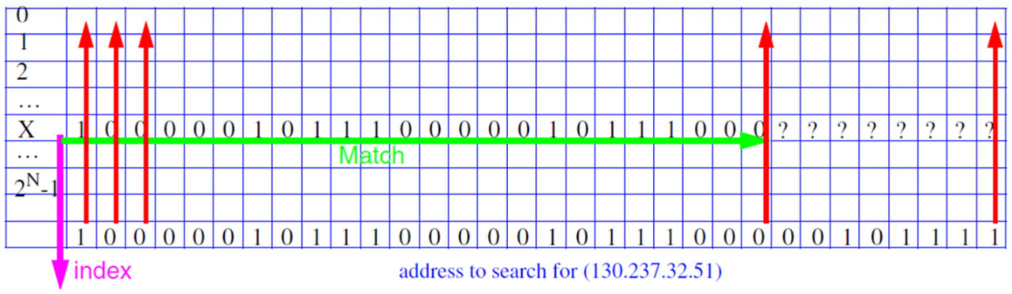
The bulk of the forwarding effort is **searching** the tables (as most of the rest of the processing is simple logical bit operations).



Routing Table Search - Classless

Match destination with longest prefixes first

- Software algorithms: tree, binary trees, tries (different data structures) [Ravikumar 2002]
- Hardware support: Content Addressable Memory (CAM)
  - Ternary CAM (TCAM) - [Renesas],[Yu 2004] - for searching and pattern matching
  - Ternary because each cell has three states: 0, 1, or ? (aka "don't care")
  - Performance an associate search (i.e., in **parallel**), returns the index (X) of the first match
  - 4-8 ns/search (i.e., 125M - 250M lookups per second)



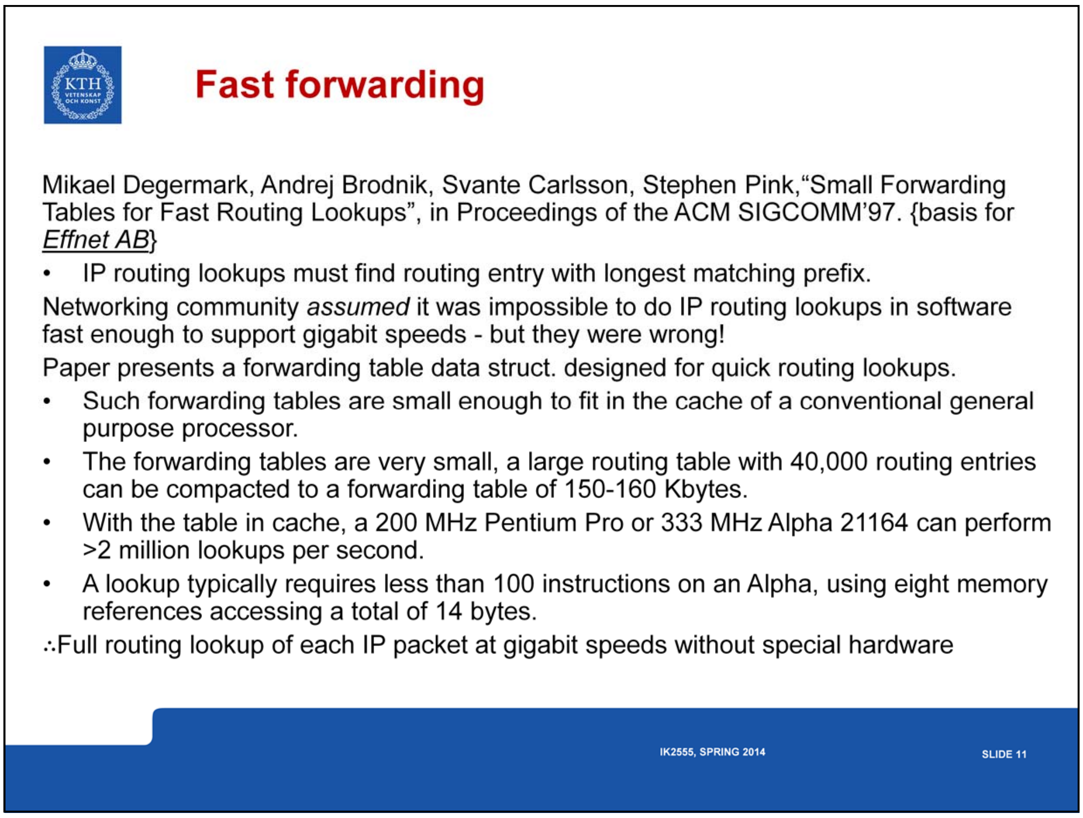
Note that the entries in the table have to be sorted - to achieve the longest match being at the smallest index.

IK2555, SPRING 2014	SLIDE 10

Renesas Technology Corp. TCAM description *http://www.renesas.com/fmwk.jsp?cnt=tcam_series_landing.jsp&fp=/applications/network/network_memory/tcam/*

Fany Yu, Randy H. Katz, and T. V. Lakshman, "Gigabit Rate Multiple-Pattern Matching with TCAM", University of California at Berkeley, Computer Science department, January 2004, *http://sahara.cs.berkeley.edu/jan2004-retreat/slides/Fang_retreat.ppt*

V. C. Ravikumar, R. Mahapatra, and J. C. Liu, 'Modified LC-trie based efficient routing lookup', presented at the 10th IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunications Systems, 2002. MASCOTS 2002, 2002, pp. 177–182, DOI:10.1109/MASCOT.2002.1167075 *http://faculty.cs.tamu.edu/rabi/Publications/Mascot-final-proceeding.pdf*



Fast forwarding

Mikael Degermark, Andrej Brodnik, Svante Carlsson, Stephen Pink,“Small Forwarding Tables for Fast Routing Lookups”, in Proceedings of the ACM SIGCOMM’97. {basis for *Effnet AB*}

- IP routing lookups must find routing entry with longest matching prefix.

Networking community *assumed* it was impossible to do IP routing lookups in software fast enough to support gigabit speeds - but they were wrong!

Paper presents a forwarding table data struct. designed for quick routing lookups.

- Such forwarding tables are small enough to fit in the cache of a conventional general purpose processor.
- The forwarding tables are very small, a large routing table with 40,000 routing entries can be compacted to a forwarding table of 150-160 Kbytes.
- With the table in cache, a 200 MHz Pentium Pro or 333 MHz Alpha 21164 can perform >2 million lookups per second.
- A lookup typically requires less than 100 instructions on an Alpha, using eight memory references accessing a total of 14 bytes.

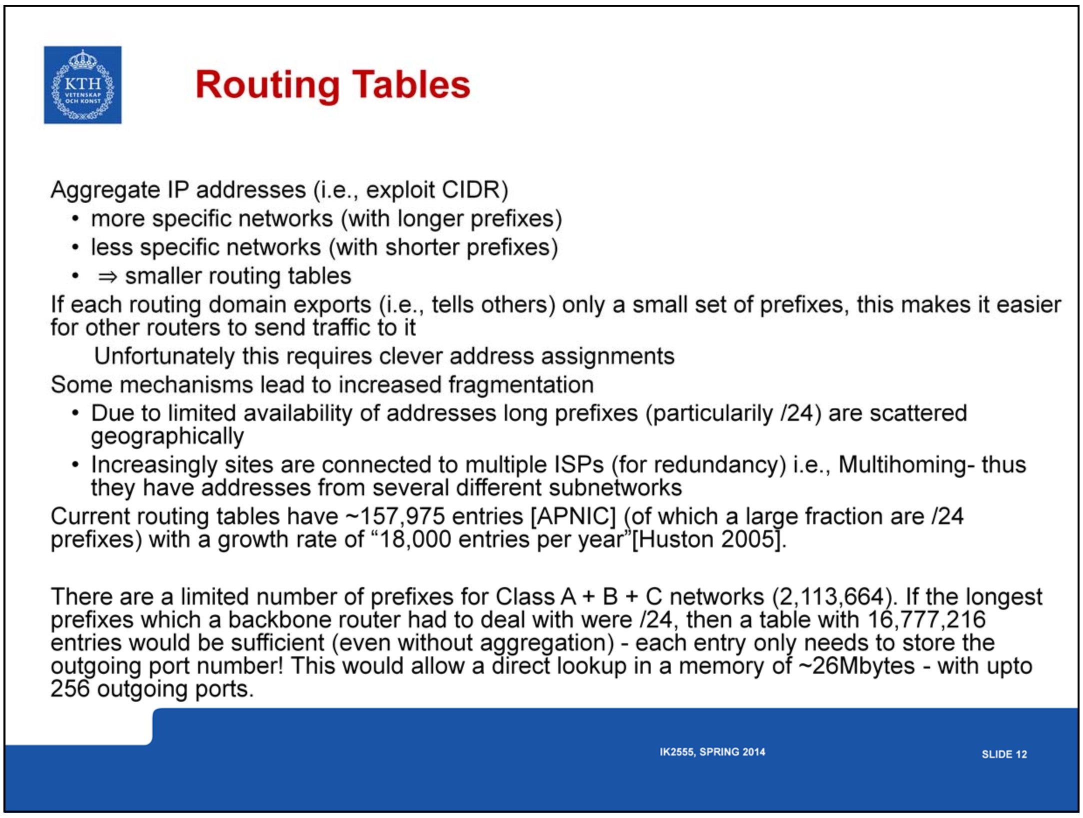
∴Full routing lookup of each IP packet at gigabit speeds without special hardware

IK2555, SPRING 2014 SLIDE 11

M. Degermark, A. Brodnik, S. Carlsson, and S. Pink, ‘Small Forwarding Tables for Fast Routing Lookups’, in *Proceedings of the ACM SIGCOMM ’97 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, New York, NY, USA, 1997, pp. 3–14 [Online]. Available: http://doi.acm.org/10.1145/263105.263133

M. Degermark, A. Brodnik, S. Carlsson, and S. Pink, ‘Small Forwarding Tables for Fast Routing Lookups’, *SIGCOMM Comput. Commun. Rev.*, vol. 27, no. 4, pp. 3–14, Oct. 1997. DOI:10.1145/263109.263133

ftp://cdt.luth.se/micke/sigcomm97-lookup.ps.Z



Routing Tables

Aggregate IP addresses (i.e., exploit CIDR)
- more specific networks (with longer prefixes)
- less specific networks (with shorter prefixes)
- ⇒ smaller routing tables

If each routing domain exports (i.e., tells others) only a small set of prefixes, this makes it easier for other routers to send traffic to it

Unfortunately this requires clever address assignments

Some mechanisms lead to increased fragmentation
- Due to limited availability of addresses long prefixes (particularily /24) are scattered geographically
- Increasingly sites are connected to multiple ISPs (for redundancy) i.e., Multihoming- thus they have addresses from several different subnetworks

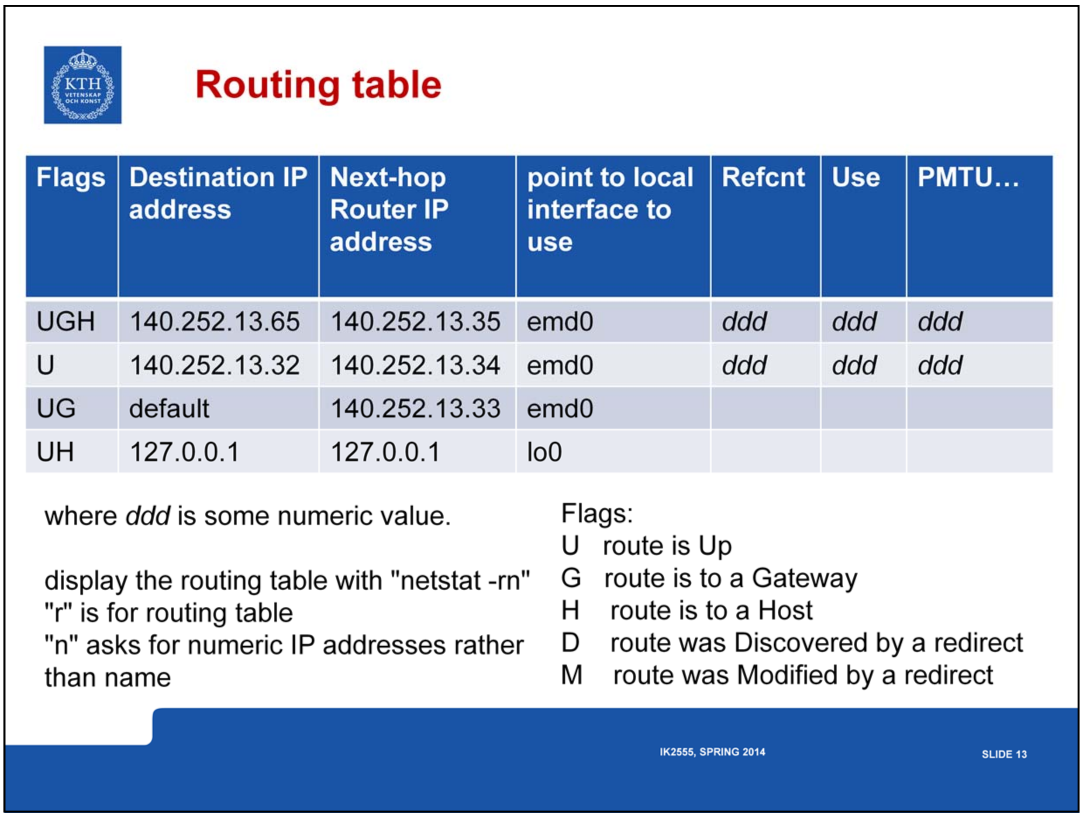
Current routing tables have ~157,975 entries [APNIC] (of which a large fraction are /24 prefixes) with a growth rate of “18,000 entries per year”[Huston 2005].

There are a limited number of prefixes for Class A + B + C networks (2,113,664). If the longest prefixes which a backbone router had to deal with were /24, then a table with 16,777,216 entries would be sufficient (even without aggregation) - each entry only needs to store the outgoing port number! This would allow a direct lookup in a memory of ~26Mbytes - with upto 256 outgoing ports.

IK2555, SPRING 2014 SLIDE 12

APNIC, Routing Table Report 04:00 +10GMT Sat 19 Mar, 2005, North American Network Operators Group, Weekly Routing Table Report, From: Routing Table Analysis, Mar 18 13:10:37 2005, "This is an automated weekly mailing describing the state of the Internet Routing Table as seen from APNIC’s router in Japan. Daily listings are sent to bgp-stats@lists.apnic.net" *http://www.merit.edu/mail.archives/nanog/2005-03/msg00401.html*

Geoff Huston, Routing Table Status Report, Policy SIG, APNIC19, Kyoto, Japan, Feb 24 2005



Routing table

Flags	Destination IP address	Next-hop Router IP address	point to local interface to use	Refcnt	Use	PMTU…
UGH	140.252.13.65	140.252.13.35	emd0	*ddd*	*ddd*	*ddd*
U	140.252.13.32	140.252.13.34	emd0	*ddd*	*ddd*	*ddd*
UG	default	140.252.13.33	emd0			
UH	127.0.0.1	127.0.0.1	lo0			

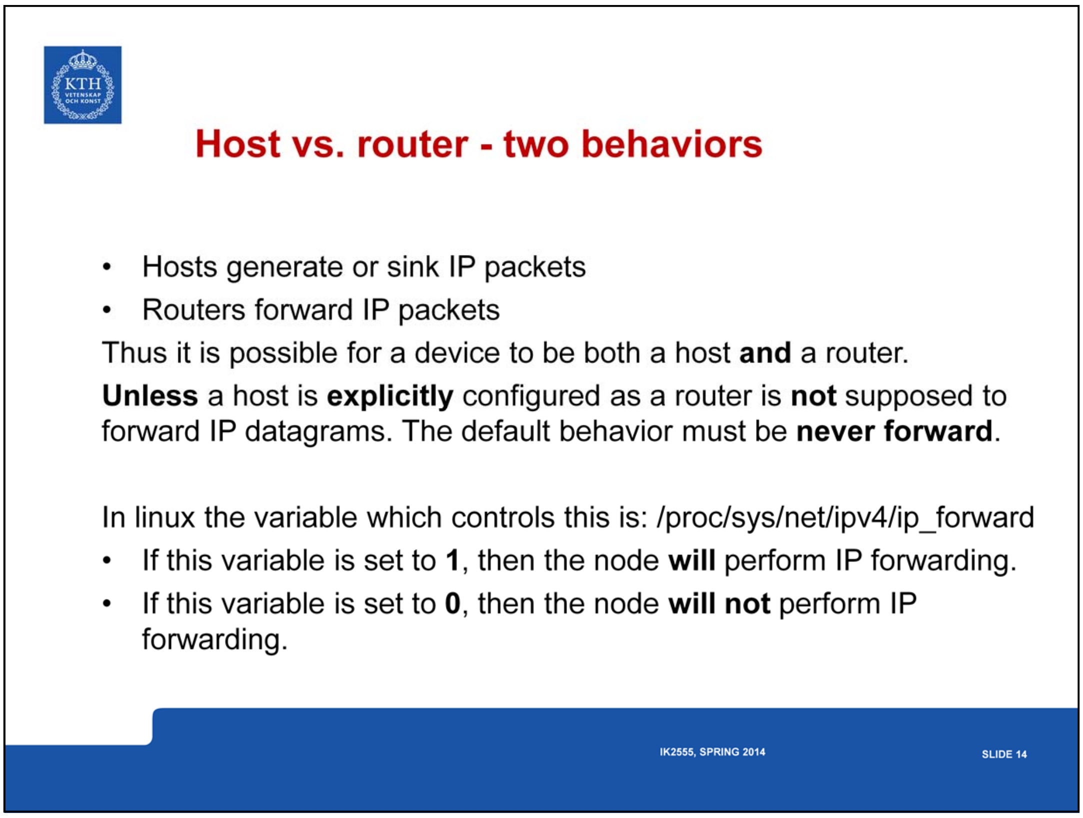
where *ddd* is some numeric value.

display the routing table with "netstat -rn"
"r" is for routing table
"n" asks for numeric IP addresses rather than name

Flags:

U	route is Up
G	route is to a Gateway
H	route is to a Host
D	route was Discovered by a redirect
M	route was Modified by a redirect

IK2555, SPRING 2014 SLIDE 13



Host vs. router - two behaviors

- Hosts generate or sink IP packets
- Routers forward IP packets

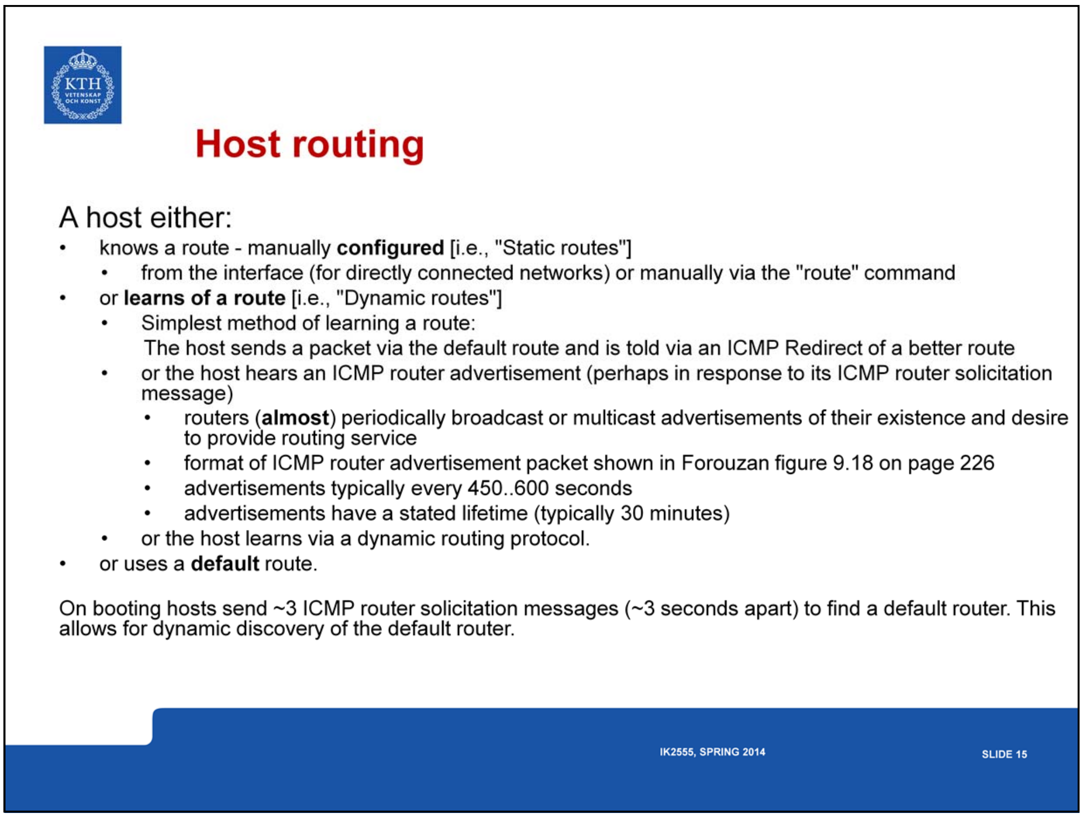
Thus it is possible for a device to be both a host **and** a router.

Unless a host is **explicitly** configured as a router is **not** supposed to forward IP datagrams. The default behavior must be **never forward**.

In linux the variable which controls this is: /proc/sys/net/ipv4/ip_forward

- If this variable is set to **1**, then the node **will** perform IP forwarding.
- If this variable is set to **0**, then the node **will not** perform IP forwarding.

IK2555, SPRING 2014 SLIDE 14



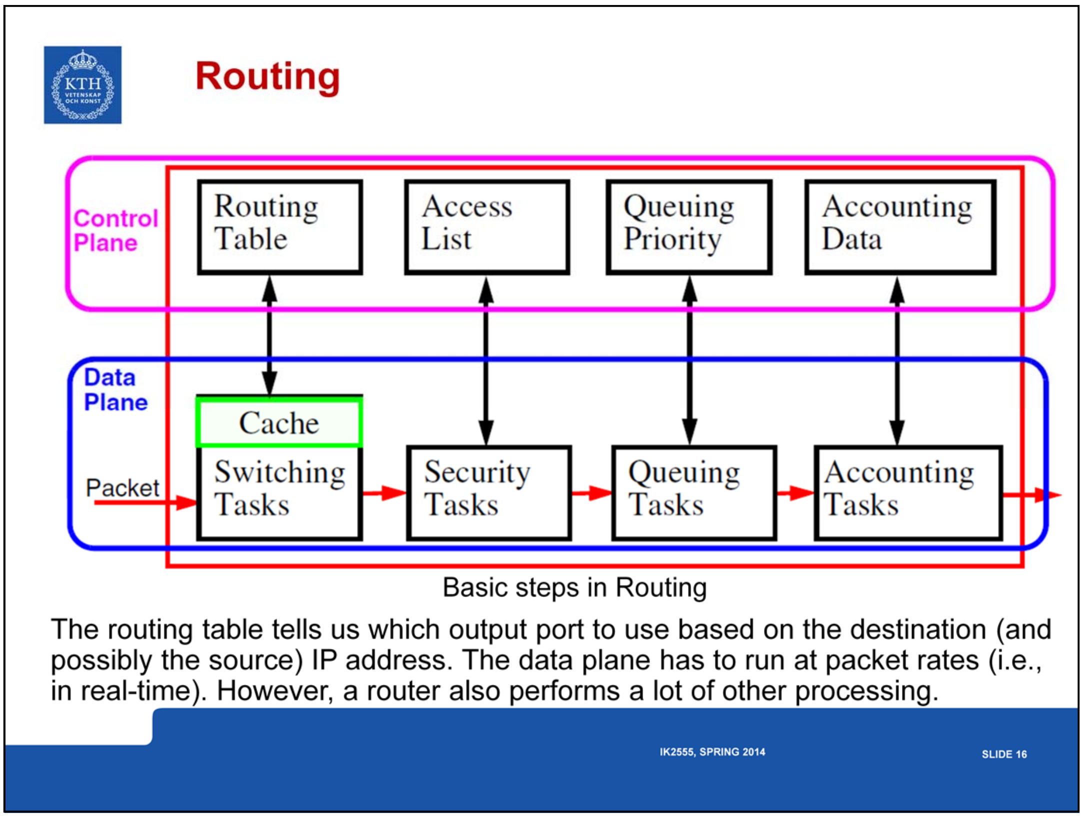
Host routing

A host either:

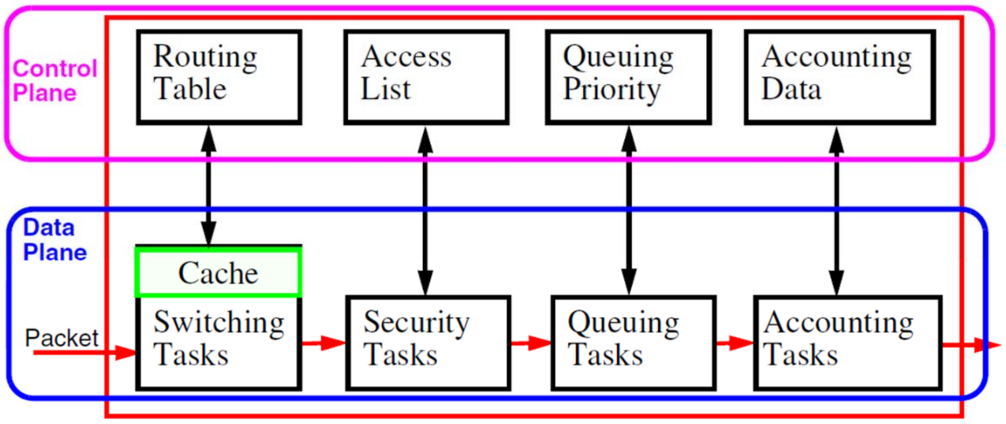
- knows a route - manually **configured** [i.e., "Static routes"]
  - from the interface (for directly connected networks) or manually via the "route" command
- or **learns of a route** [i.e., "Dynamic routes"]
  - Simplest method of learning a route:
    The host sends a packet via the default route and is told via an ICMP Redirect of a better route
  - or the host hears an ICMP router advertisement (perhaps in response to its ICMP router solicitation message)
    - routers (**almost**) periodically broadcast or multicast advertisements of their existence and desire to provide routing service
    - format of ICMP router advertisement packet shown in Forouzan figure 9.18 on page 226
    - advertisements typically every 450..600 seconds
    - advertisements have a stated lifetime (typically 30 minutes)
  - or the host learns via a dynamic routing protocol.
- or uses a **default** route.

On booting hosts send ~3 ICMP router solicitation messages (~3 seconds apart) to find a default router. This allows for dynamic discovery of the default router.

IK2555, SPRING 2014 SLIDE 15



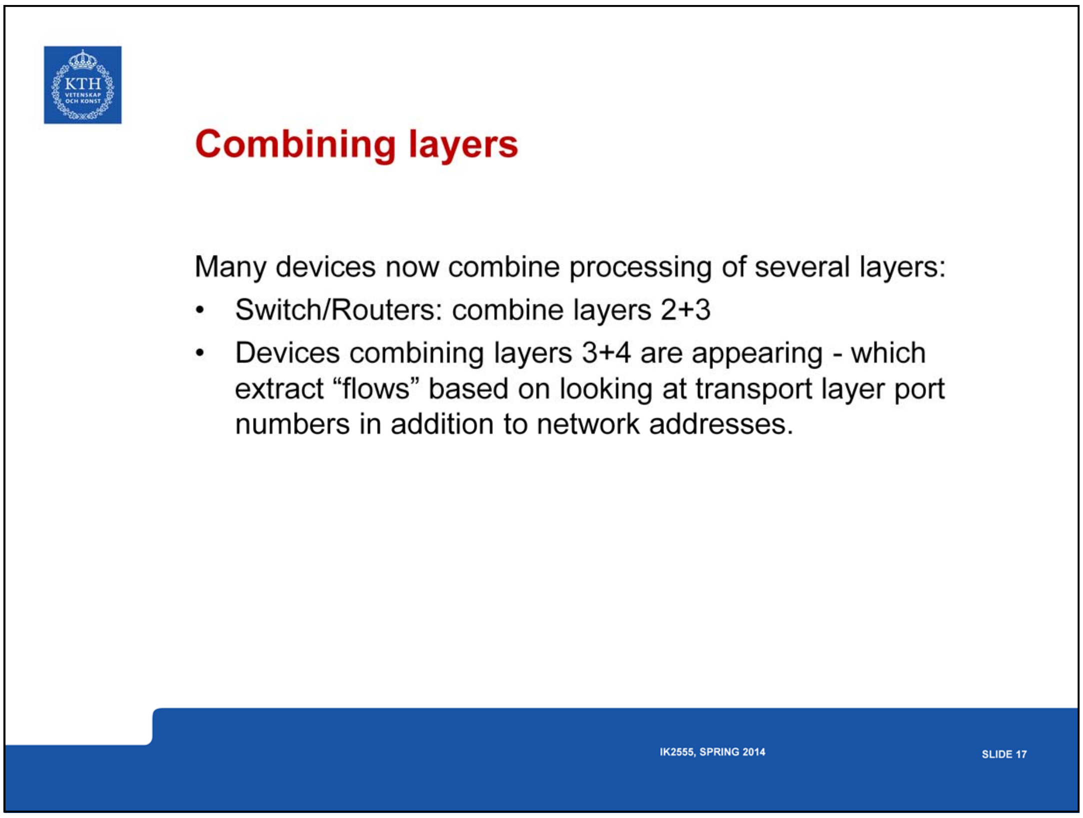
Routing



Basic steps in Routing

The routing table tells us which output port to use based on the destination (and possibly the source) IP address. The data plane has to run at packet rates (i.e., in real-time). However, a router also performs a lot of other processing.

IK2555, SPRING 2014   SLIDE 16

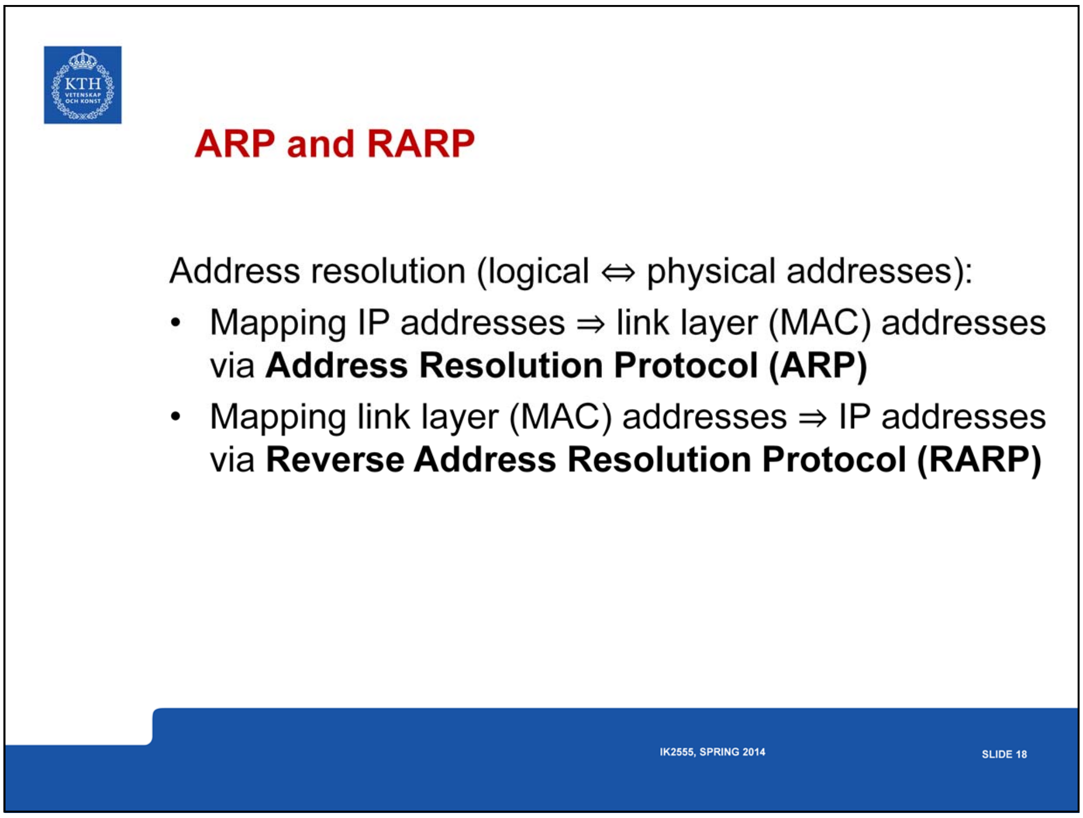


Combining layers

Many devices now combine processing of several layers:

- Switch/Routers: combine layers 2+3
- Devices combining layers 3+4 are appearing - which extract “flows” based on looking at transport layer port numbers in addition to network addresses.

IK2555, SPRING 2014 SLIDE 17

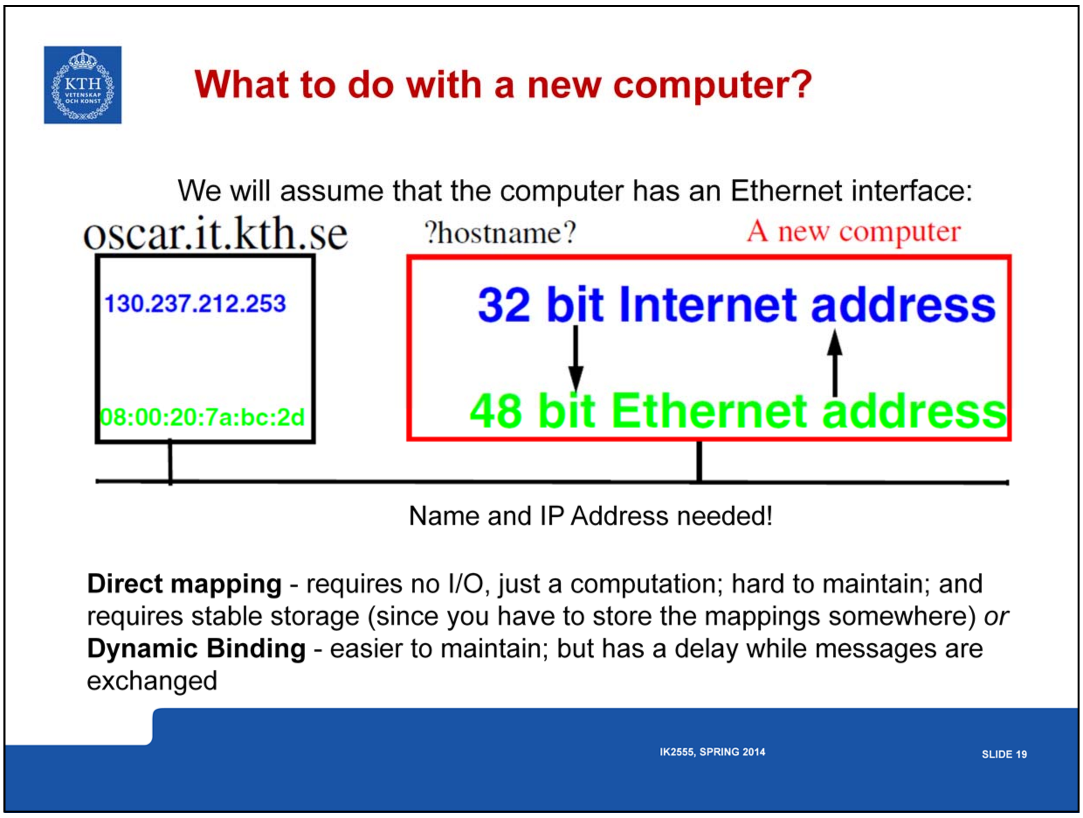


ARP and RARP

Address resolution (logical ⇔ physical addresses):

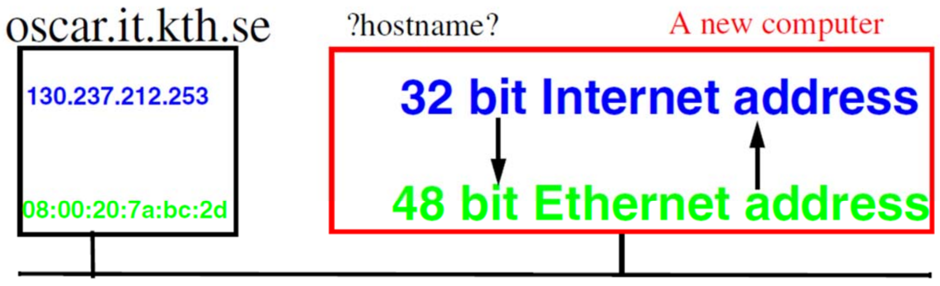
- Mapping IP addresses ⇒ link layer (MAC) addresses via **Address Resolution Protocol (ARP)**
- Mapping link layer (MAC) addresses ⇒ IP addresses via **Reverse Address Resolution Protocol (RARP)**

IK2555, SPRING 2014 SLIDE 18



What to do with a new computer?

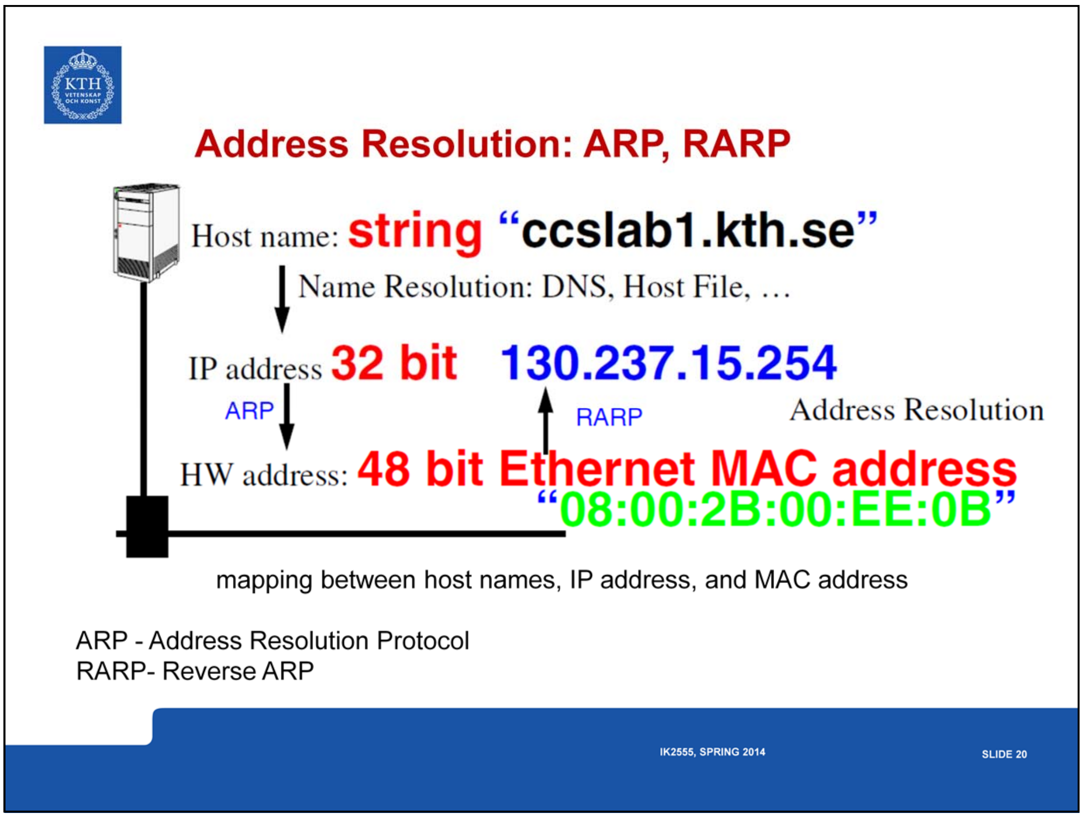
We will assume that the computer has an Ethernet interface:



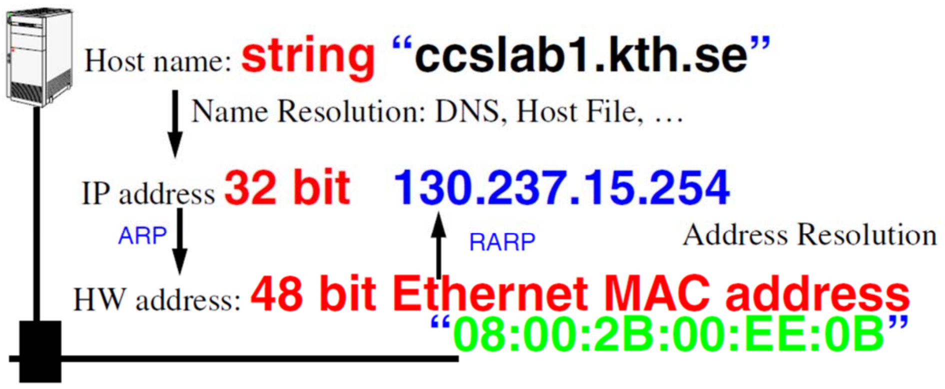
Name and IP Address needed!

Direct mapping - requires no I/O, just a computation; hard to maintain; and requires stable storage (since you have to store the mappings somewhere) *or* **Dynamic Binding** - easier to maintain; but has a delay while messages are exchanged

IK2555, SPRING 2014 SLIDE 19



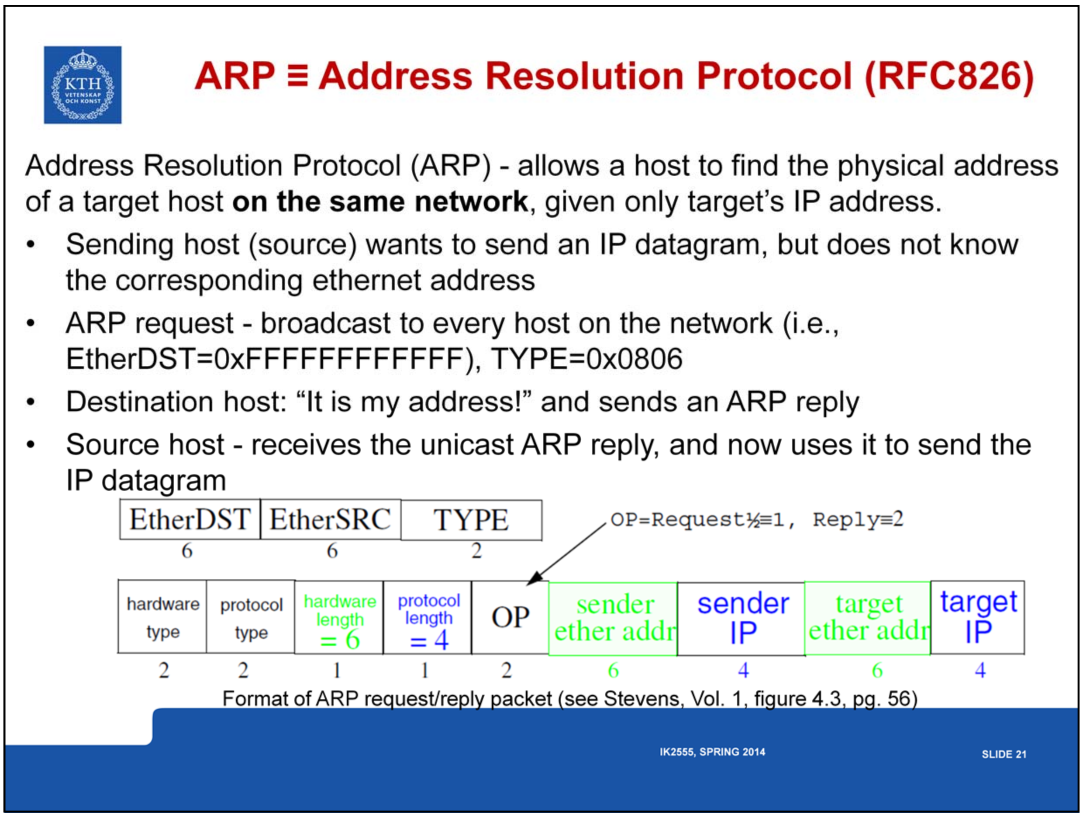
Address Resolution: ARP, RARP


Host name: string “ccslab1.kth.se”
Name Resolution: DNS, Host File, …
IP address 32 bit 130.237.15.254
ARP
RARP
Address Resolution
HW address: 48 bit Ethernet MAC address
“08:00:2B:00:EE:0B”

mapping between host names, IP address, and MAC address

ARP - Address Resolution Protocol
RARP- Reverse ARP

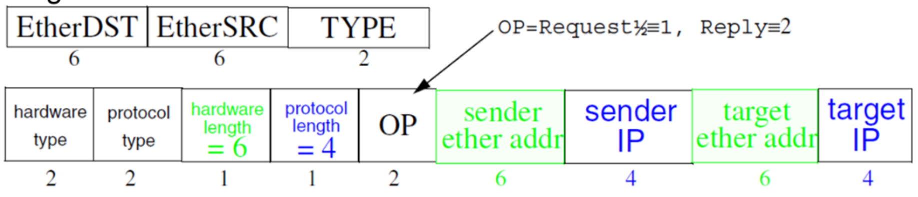
IK2555, SPRING 2014 SLIDE 20



ARP ≡ Address Resolution Protocol (RFC826)

Address Resolution Protocol (ARP) - allows a host to find the physical address of a target host **on the same network**, given only target's IP address.

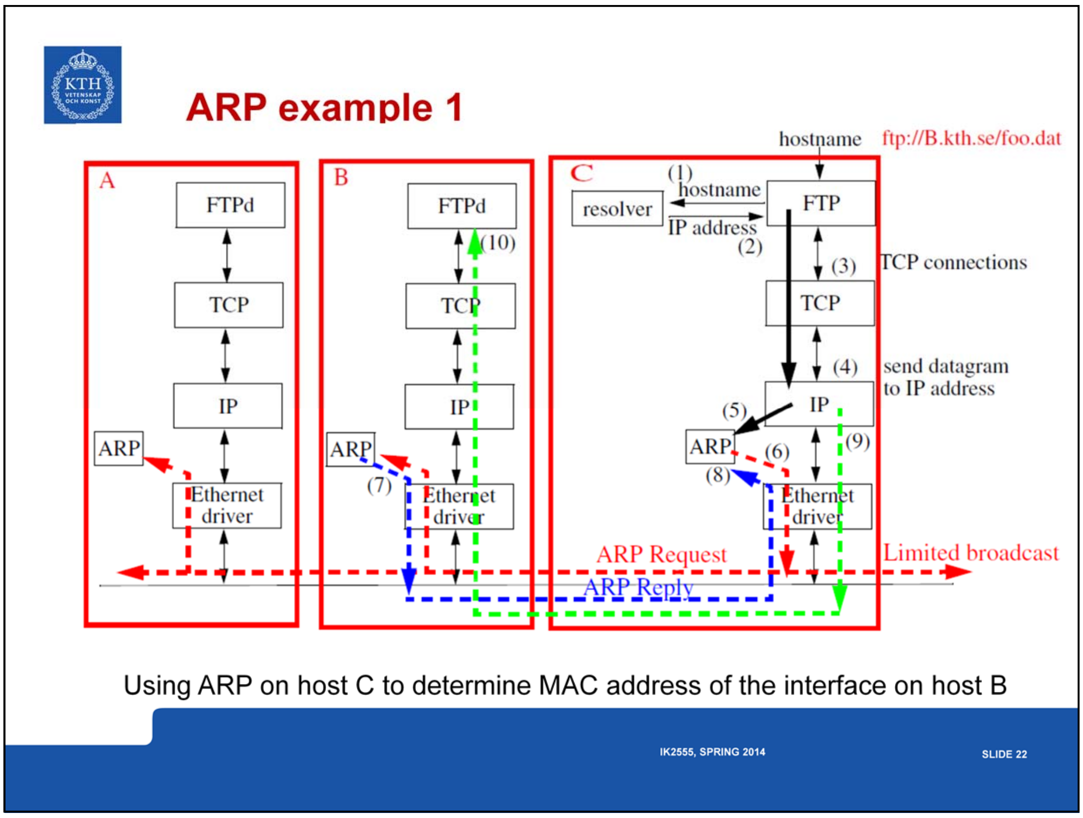
- Sending host (source) wants to send an IP datagram, but does not know the corresponding ethernet address
- ARP request - broadcast to every host on the network (i.e., EtherDST=0xFFFFFFFFFFFF), TYPE=0x0806
- Destination host: "It is my address!" and sends an ARP reply
- Source host - receives the unicast ARP reply, and now uses it to send the IP datagram



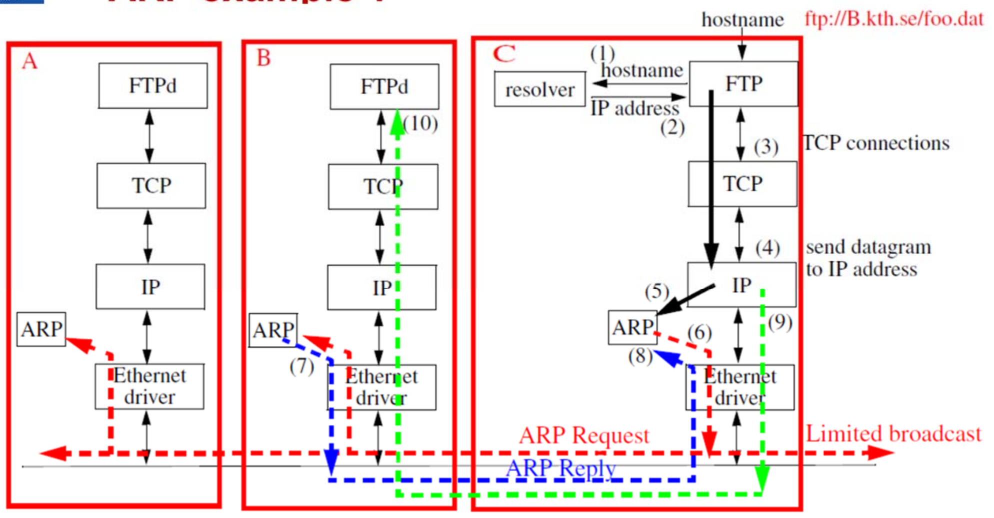
Format of ARP request/reply packet (see Stevens, Vol. 1, figure 4.3, pg. 56)

IK2555, SPRING 2014   SLIDE 21

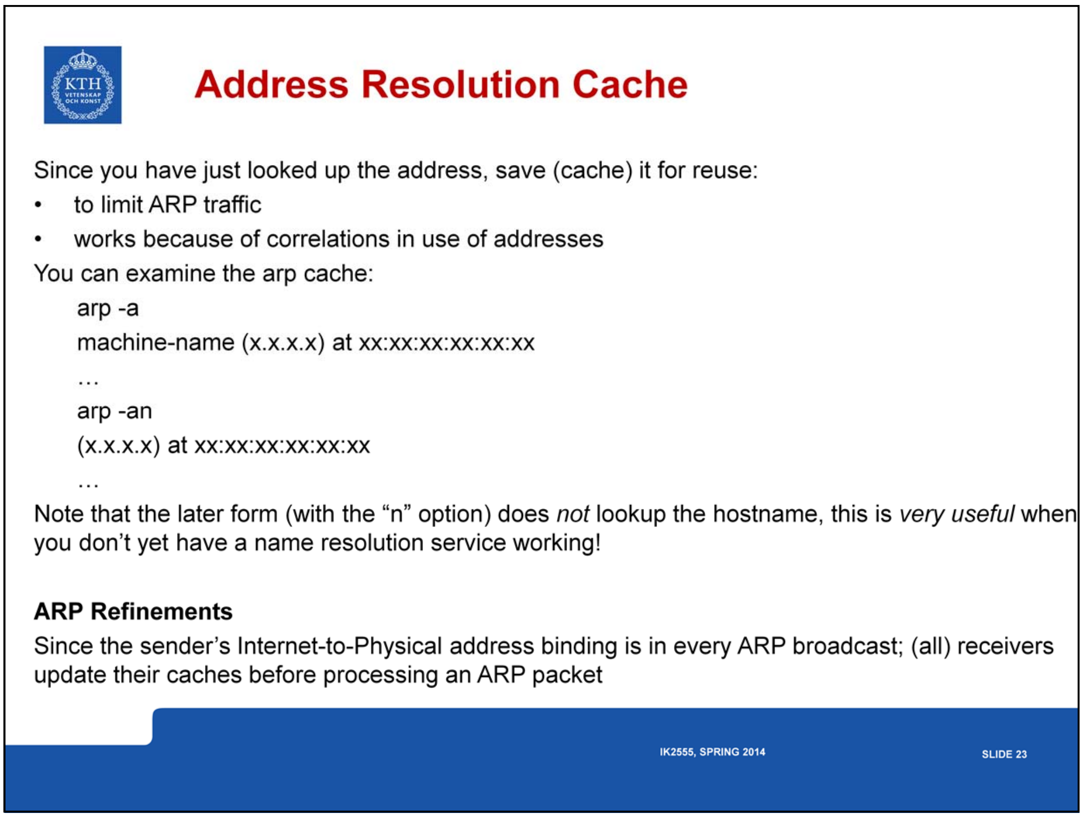
D. Plummer, 'Ethernet Address Resolution Protocol: Or Converting Network Protocol Addresses to 48.bit Ethernet Address for Transmission on Ethernet Hardware', *Internet Request for Comments*, vol. RFC 826 (INTERNET STANDARD), Nov. 1982 [Online]. Available: http://www.rfc-editor.org/rfc/rfc826.txt



ARP example 1



Using ARP on host C to determine MAC address of the interface on host B



Address Resolution Cache

Since you have just looked up the address, save (cache) it for reuse:

- to limit ARP traffic
- works because of correlations in use of addresses

You can examine the arp cache:

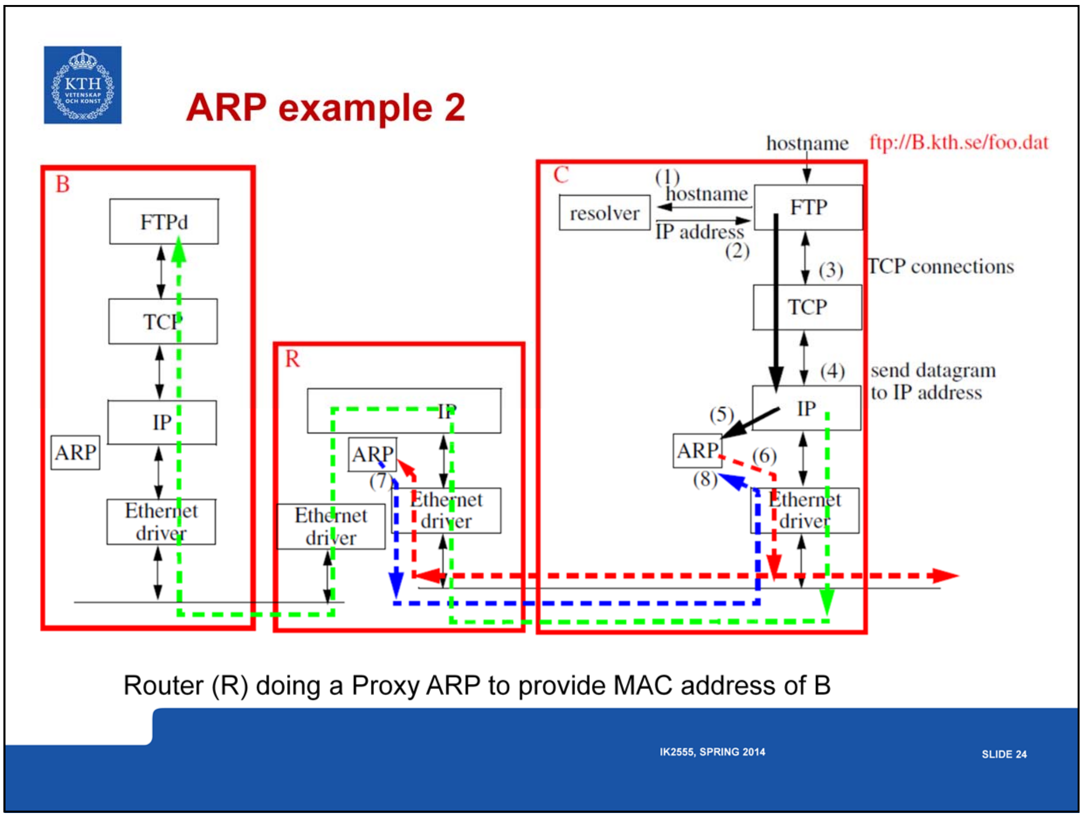
```
arp -a
machine-name (x.x.x.x) at xx:xx:xx:xx:xx:xx
…
arp -an
(x.x.x.x) at xx:xx:xx:xx:xx:xx
…
```

Note that the later form (with the "n" option) does *not* lookup the hostname, this is *very useful* when you don't yet have a name resolution service working!

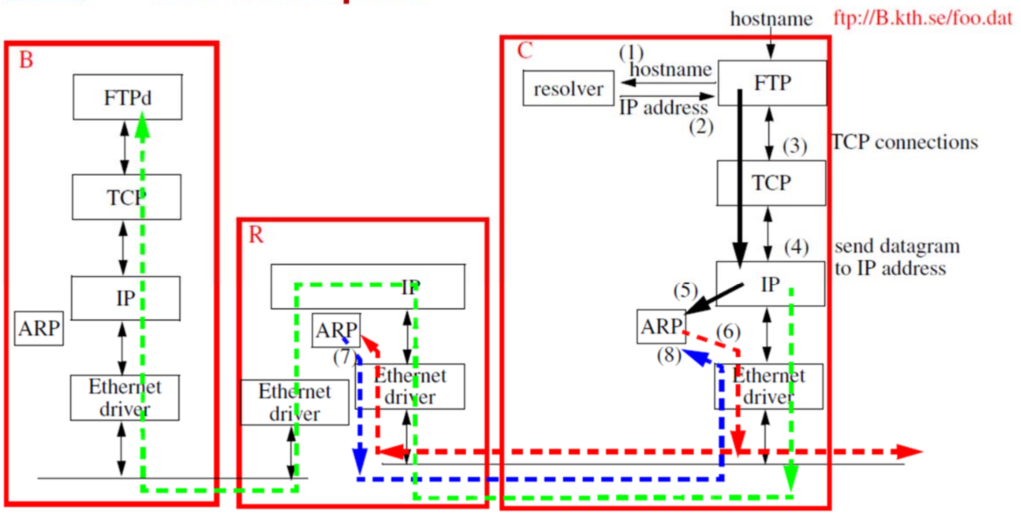
ARP Refinements

Since the sender's Internet-to-Physical address binding is in every ARP broadcast; (all) receivers update their caches before processing an ARP packet

IK2555, SPRING 2014 SLIDE 23

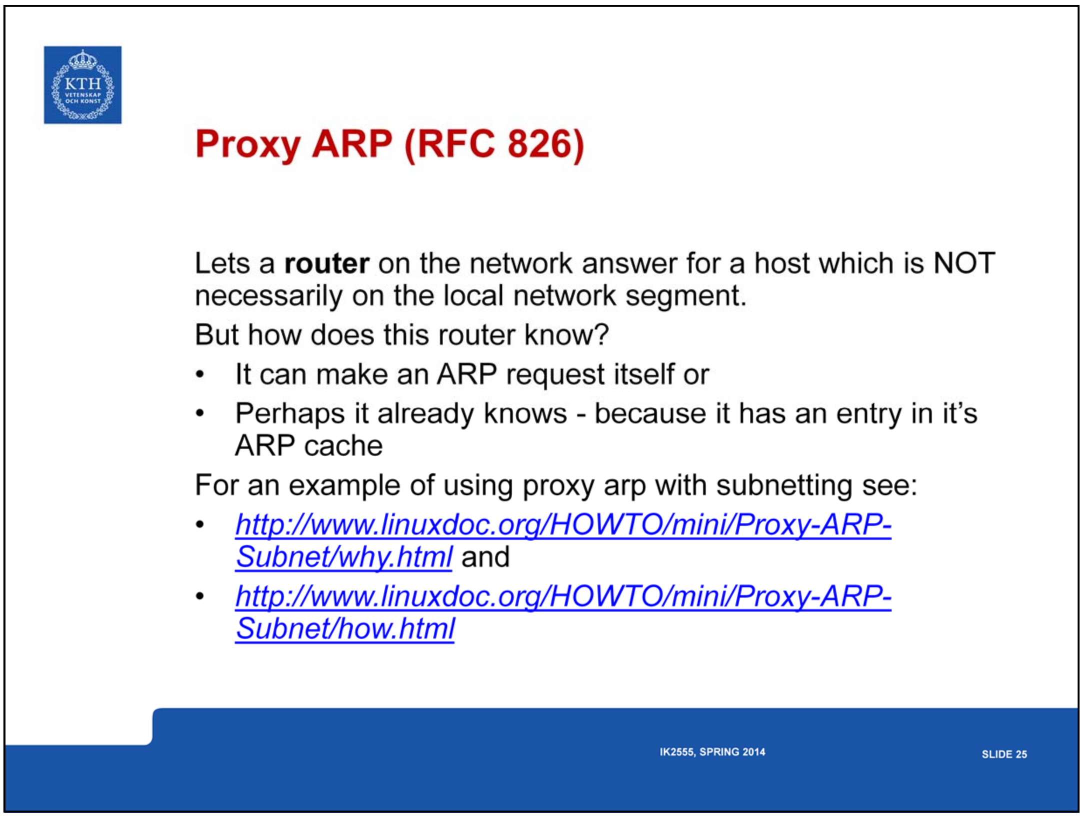


ARP example 2



Router (R) doing a Proxy ARP to provide MAC address of B

IK2555, SPRING 2014 SLIDE 24



Proxy ARP (RFC 826)

Lets a **router** on the network answer for a host which is NOT necessarily on the local network segment.

But how does this router know?

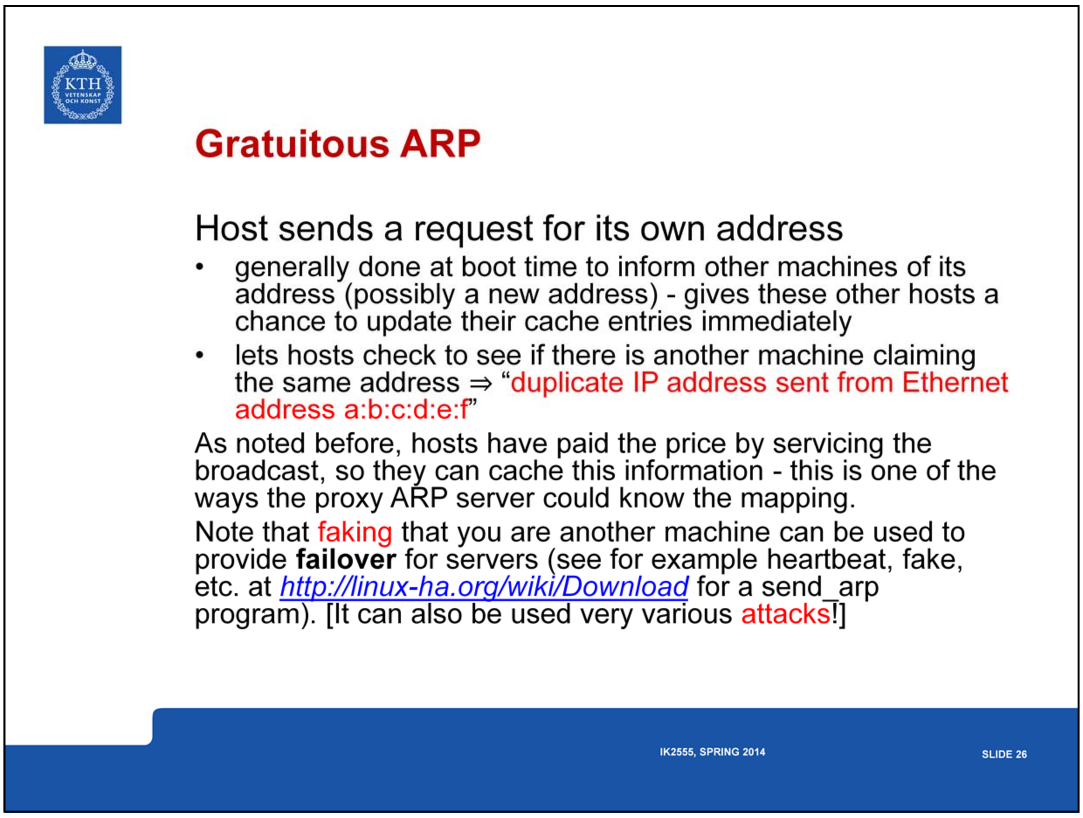
- It can make an ARP request itself or
- Perhaps it already knows - because it has an entry in it's ARP cache

For an example of using proxy arp with subnetting see:

- *http://www.linuxdoc.org/HOWTO/mini/Proxy-ARP-Subnet/why.html* and
- *http://www.linuxdoc.org/HOWTO/mini/Proxy-ARP-Subnet/how.html*

IK2555, SPRING 2014 SLIDE 25

D. Plummer, 'Ethernet Address Resolution Protocol: Or Converting Network Protocol Addresses to 48.bit Ethernet Address for Transmission on Ethernet Hardware', *Internet Request for Comments*, vol. RFC 826 (INTERNET STANDARD), Nov. 1982 [Online]. Available: http://www.rfc-editor.org/rfc/rfc826.txt



Gratuitous ARP

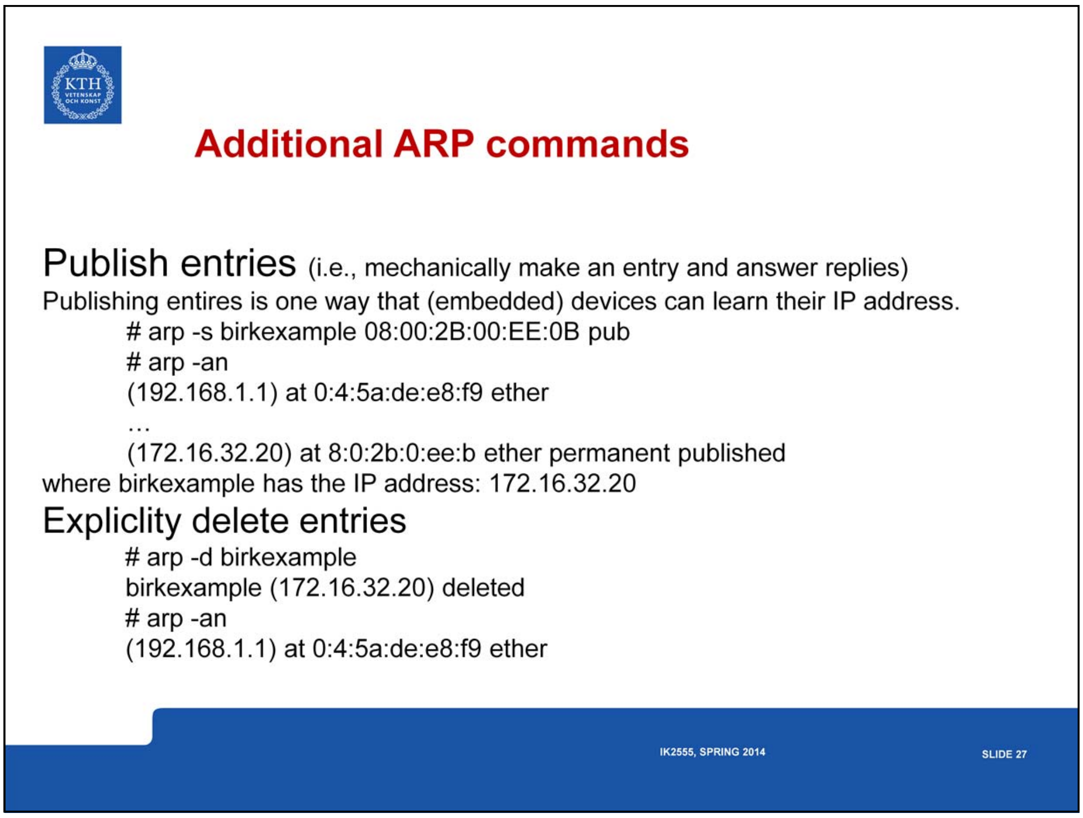
Host sends a request for its own address

- generally done at boot time to inform other machines of its address (possibly a new address) - gives these other hosts a chance to update their cache entries immediately
- lets hosts check to see if there is another machine claiming the same address ⇒ "duplicate IP address sent from Ethernet address a:b:c:d:e:f"

As noted before, hosts have paid the price by servicing the broadcast, so they can cache this information - this is one of the ways the proxy ARP server could know the mapping.

Note that faking that you are another machine can be used to provide **failover** for servers (see for example heartbeat, fake, etc. at *http://linux-ha.org/wiki/Download* for a send_arp program). [It can also be used very various attacks!]

IK2555, SPRING 2014 SLIDE 26



Additional ARP commands

Publish entries (i.e., mechanically make an entry and answer replies)

Publishing entires is one way that (embedded) devices can learn their IP address.

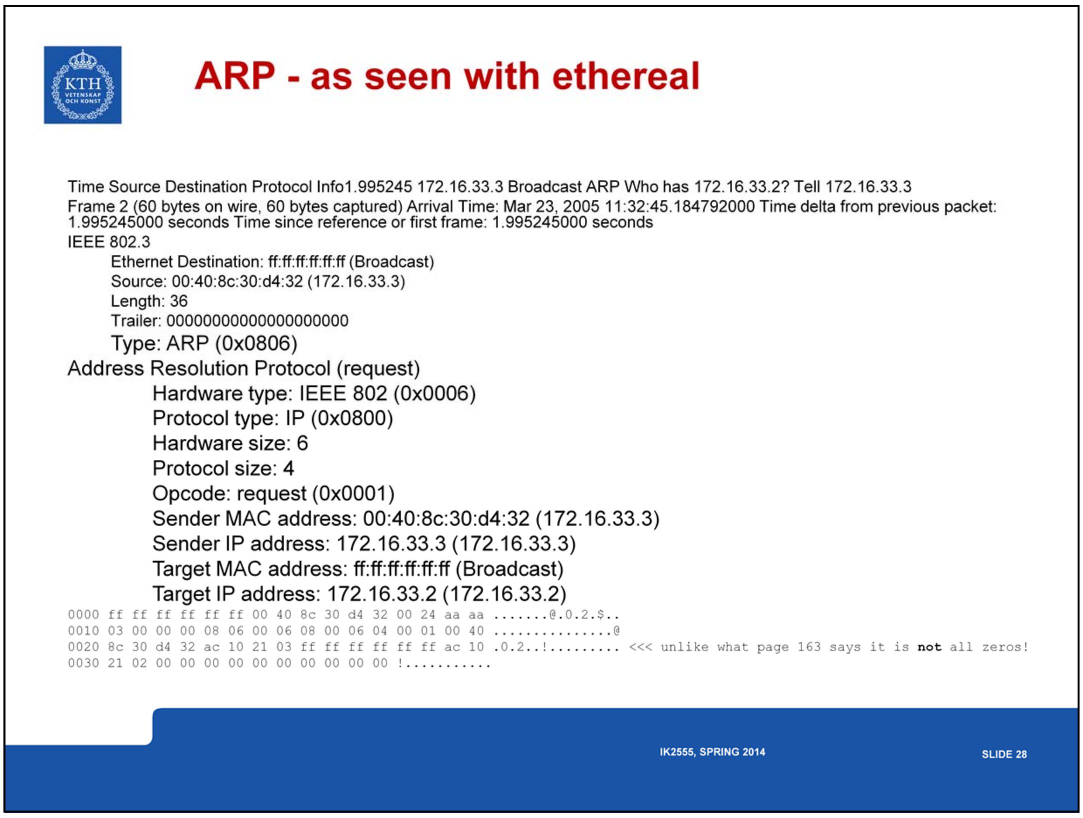
```
# arp -s birkexample 08:00:2B:00:EE:0B pub
# arp -an
(192.168.1.1) at 0:4:5a:de:e8:f9 ether
…
(172.16.32.20) at 8:0:2b:0:ee:b ether permanent published
```

where birkexample has the IP address: 172.16.32.20

Expliclity delete entries

```
# arp -d birkexample
birkexample (172.16.32.20) deleted
# arp -an
(192.168.1.1) at 0:4:5a:de:e8:f9 ether
```

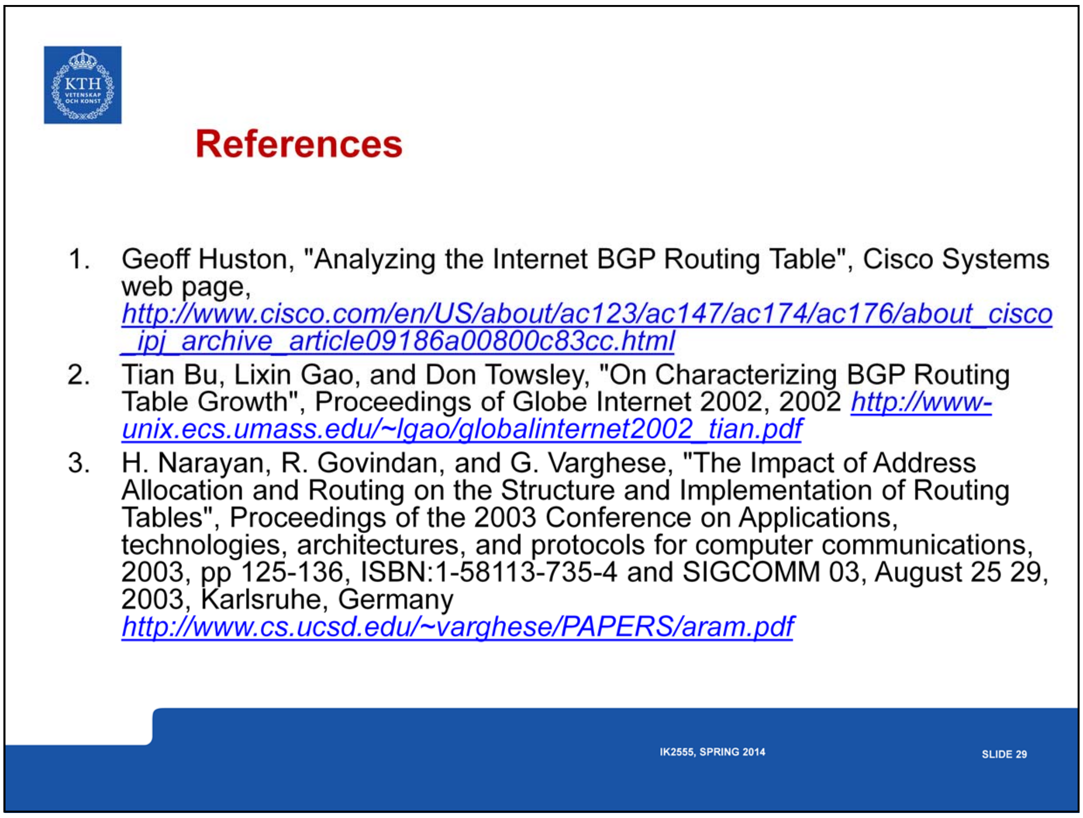
IK2555, SPRING 2014   SLIDE 27



ARP - as seen with ethereal

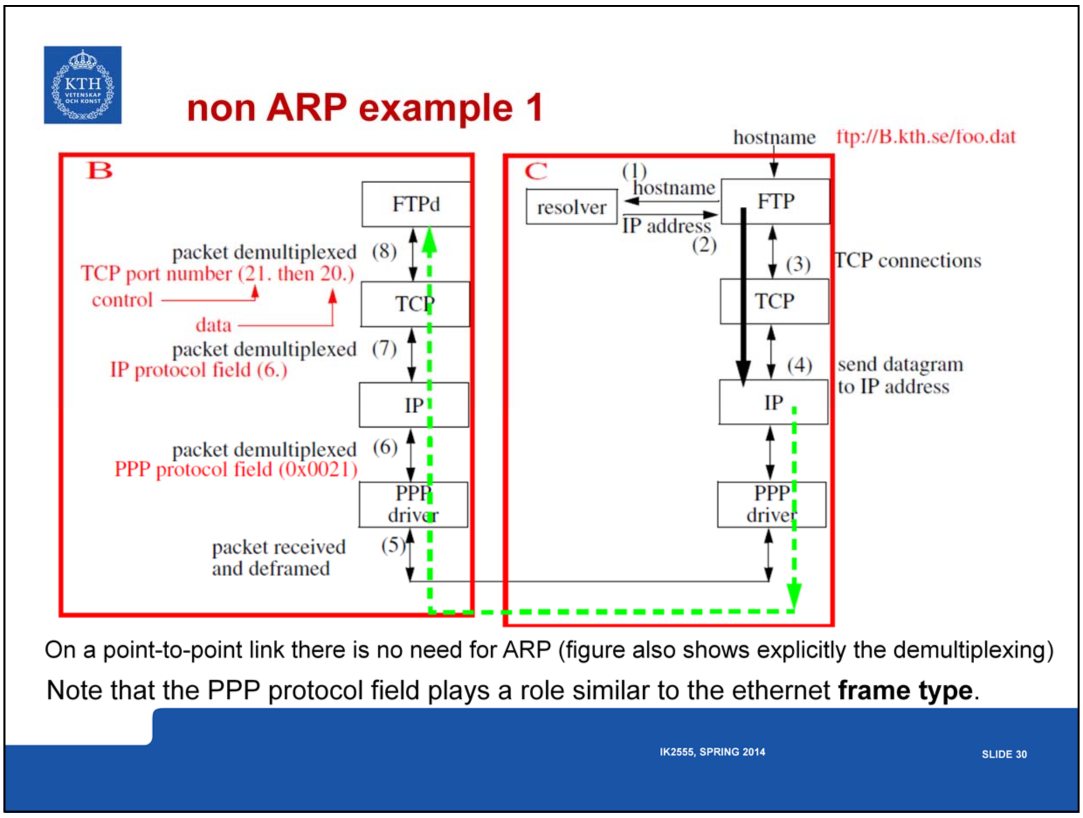
```
Time Source Destination Protocol Info1.995245 172.16.33.3 Broadcast ARP Who has 172.16.33.2? Tell 172.16.33.3
Frame 2 (60 bytes on wire, 60 bytes captured) Arrival Time: Mar 23, 2005 11:32:45.184792000 Time delta from previous packet:
1.995245000 seconds Time since reference or first frame: 1.995245000 seconds
IEEE 802.3
    Ethernet Destination: ff:ff:ff:ff:ff:ff (Broadcast)
    Source: 00:40:8c:30:d4:32 (172.16.33.3)
    Length: 36
    Trailer: 00000000000000000000
    Type: ARP (0x0806)
Address Resolution Protocol (request)
        Hardware type: IEEE 802 (0x0006)
        Protocol type: IP (0x0800)
        Hardware size: 6
        Protocol size: 4
        Opcode: request (0x0001)
        Sender MAC address: 00:40:8c:30:d4:32 (172.16.33.3)
        Sender IP address: 172.16.33.3 (172.16.33.3)
        Target MAC address: ff:ff:ff:ff:ff:ff (Broadcast)
        Target IP address: 172.16.33.2 (172.16.33.2)
0000 ff ff ff ff ff ff 00 40 8c 30 d4 32 00 24 aa aa .......@.0.2.$..
0010 03 00 00 00 08 06 00 06 08 00 06 04 00 01 00 40 ...............@
0020 8c 30 d4 32 ac 10 21 03 ff ff ff ff ff ff ac 10 .0.2..!......... <<< unlike what page 163 says it is not all zeros!
0030 21 02 00 00 00 00 00 00 00 00 00 00 !...........
```

IK2555, SPRING 2014 SLIDE 28

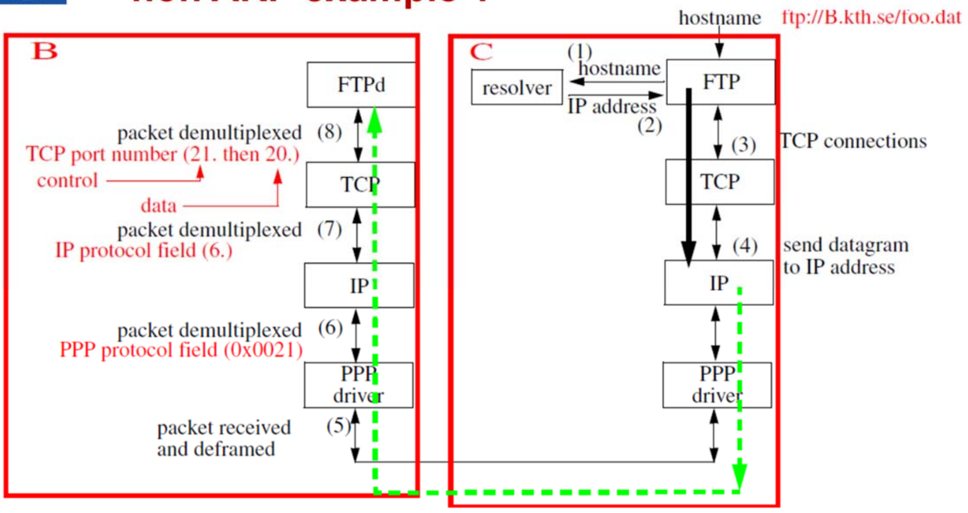


References

1. Geoff Huston, "Analyzing the Internet BGP Routing Table", Cisco Systems web page, http://www.cisco.com/en/US/about/ac123/ac147/ac174/ac176/about_cisco_ipj_archive_article09186a00800c83cc.html
2. Tian Bu, Lixin Gao, and Don Towsley, "On Characterizing BGP Routing Table Growth", Proceedings of Globe Internet 2002, 2002 http://www-unix.ecs.umass.edu/~lgao/globalinternet2002_tian.pdf
3. H. Narayan, R. Govindan, and G. Varghese, "The Impact of Address Allocation and Routing on the Structure and Implementation of Routing Tables", Proceedings of the 2003 Conference on Applications, technologies, architectures, and protocols for computer communications, 2003, pp 125-136, ISBN:1-58113-735-4 and SIGCOMM 03, August 25 29, 2003, Karlsruhe, Germany http://www.cs.ucsd.edu/~varghese/PAPERS/aram.pdf



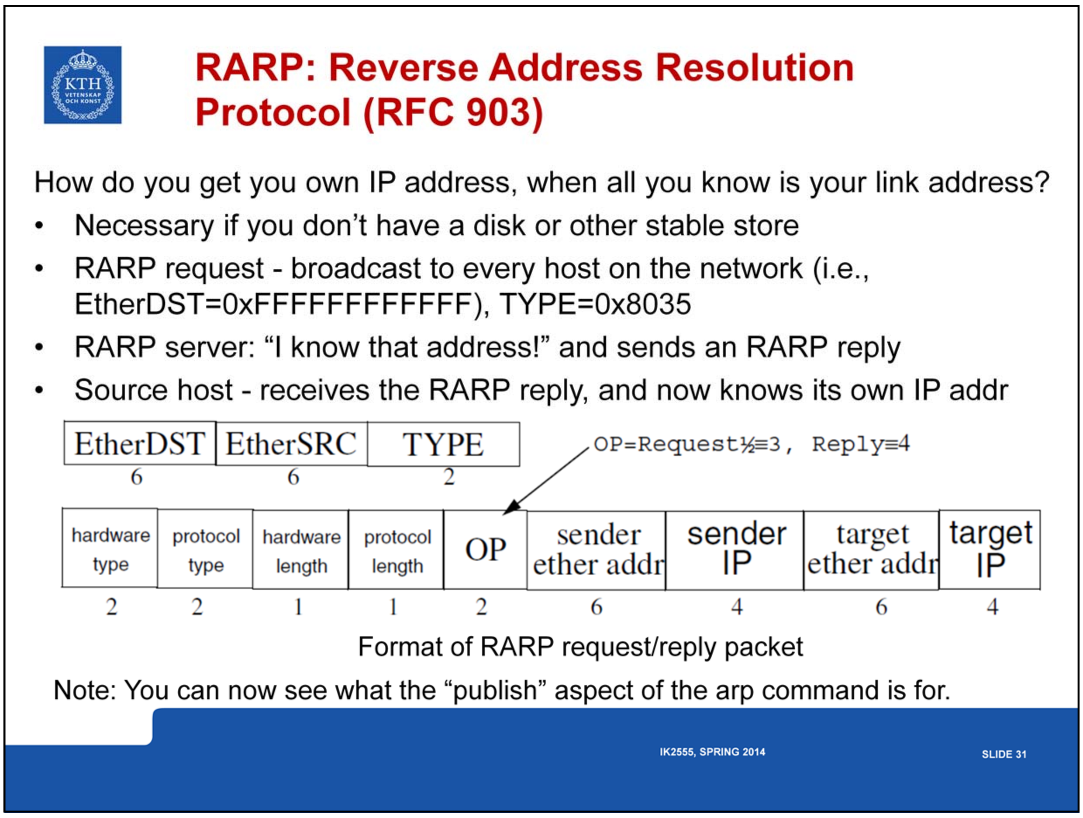
non ARP example 1



On a point-to-point link there is no need for ARP (figure also shows explicitly the demultiplexing)

Note that the PPP protocol field plays a role similar to the ethernet **frame type**.

IK2555, SPRING 2014 SLIDE 30



RARP: Reverse Address Resolution Protocol (RFC 903)

How do you get you own IP address, when all you know is your link address?

- Necessary if you don’t have a disk or other stable store
- RARP request - broadcast to every host on the network (i.e., EtherDST=0xFFFFFFFFFFFF), TYPE=0x8035
- RARP server: “I know that address!” and sends an RARP reply
- Source host - receives the RARP reply, and now knows its own IP addr

EtherDST	EtherSRC	TYPE
6	6	2

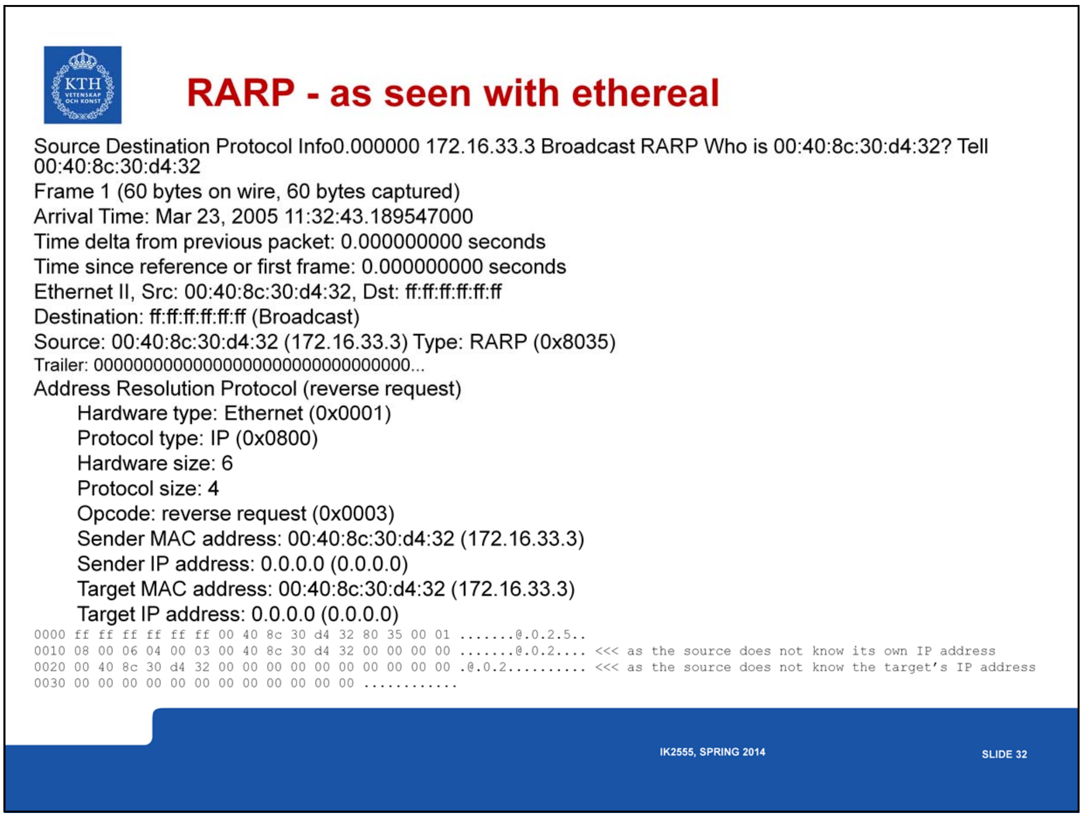
OP=Request½≡3, Reply≡4

hardware type	protocol type	hardware length	protocol length	OP	sender ether addr	sender IP	target ether addr	target IP
2	2	1	1	2	6	4	6	4

Format of RARP request/reply packet

Note: You can now see what the “publish” aspect of the arp command is for.

IK2555, SPRING 2014 SLIDE 31



RARP - as seen with ethereal

Source Destination Protocol Info0.000000 172.16.33.3 Broadcast RARP Who is 00:40:8c:30:d4:32? Tell 00:40:8c:30:d4:32

Frame 1 (60 bytes on wire, 60 bytes captured)

Arrival Time: Mar 23, 2005 11:32:43.189547000

Time delta from previous packet: 0.000000000 seconds

Time since reference or first frame: 0.000000000 seconds

Ethernet II, Src: 00:40:8c:30:d4:32, Dst: ff:ff:ff:ff:ff:ff

Destination: ff:ff:ff:ff:ff:ff (Broadcast)

Source: 00:40:8c:30:d4:32 (172.16.33.3) Type: RARP (0x8035)

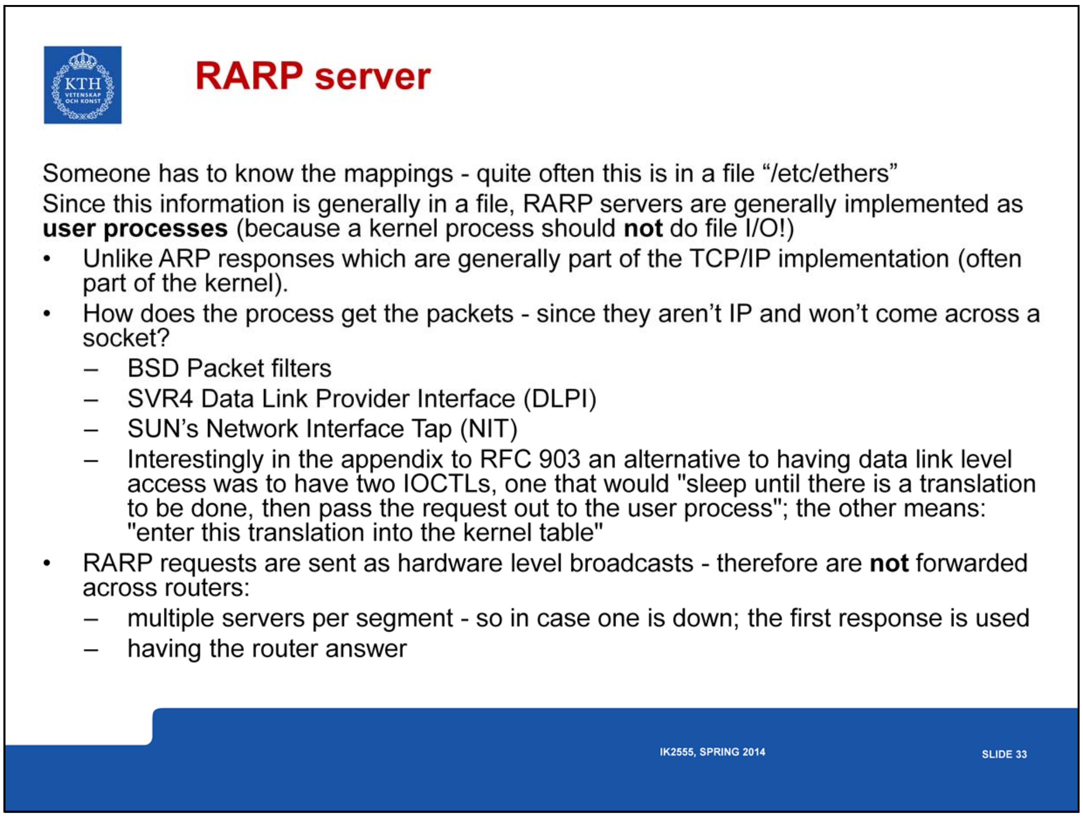
Trailer: 000000000000000000000000000000000...

Address Resolution Protocol (reverse request)

- Hardware type: Ethernet (0x0001)
- Protocol type: IP (0x0800)
- Hardware size: 6
- Protocol size: 4
- Opcode: reverse request (0x0003)
- Sender MAC address: 00:40:8c:30:d4:32 (172.16.33.3)
- Sender IP address: 0.0.0.0 (0.0.0.0)
- Target MAC address: 00:40:8c:30:d4:32 (172.16.33.3)
- Target IP address: 0.0.0.0 (0.0.0.0)

```
0000 ff ff ff ff ff ff 00 40 8c 30 d4 32 80 35 00 01 .......@.0.2.5..
0010 08 00 06 04 00 03 00 40 8c 30 d4 32 00 00 00 00 .......@.0.2.... <<< as the source does not know its own IP address
0020 00 40 8c 30 d4 32 00 00 00 00 00 00 00 00 00 00 .@.0.2.......... <<< as the source does not know the target's IP address
0030 00 00 00 00 00 00 00 00 00 00 00 00 ............
```

IK2555, SPRING 2014 SLIDE 32



RARP server

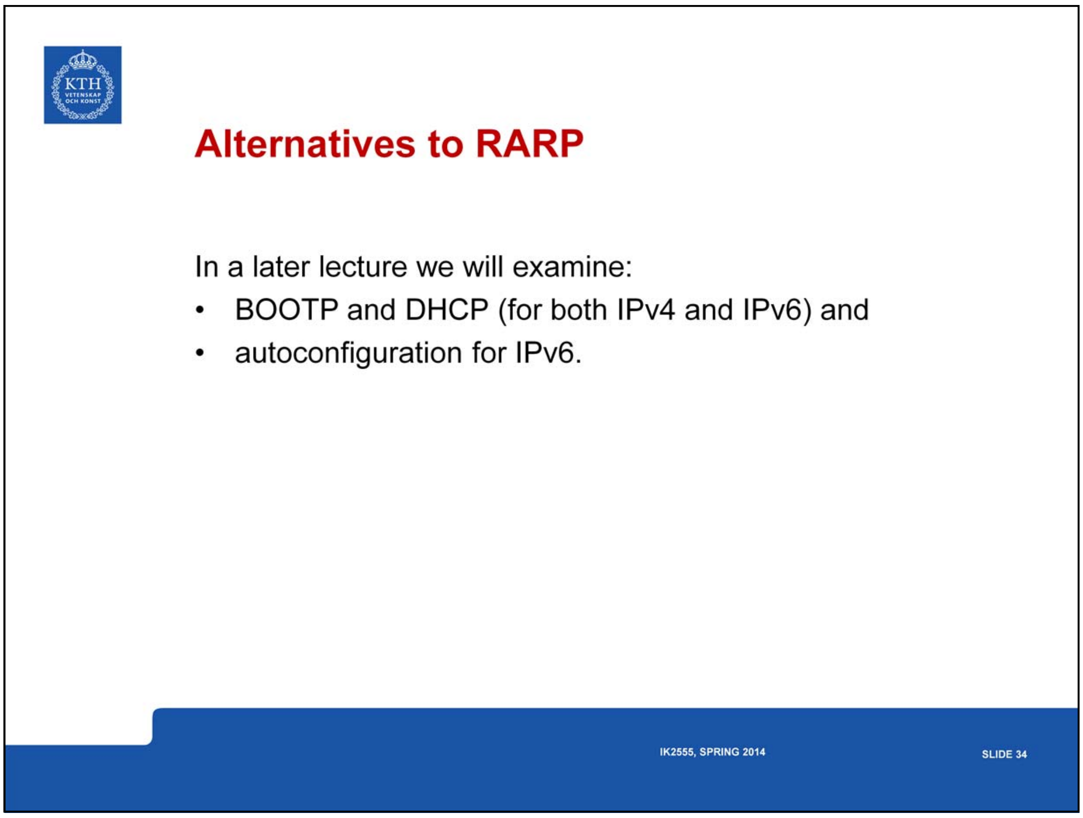
Someone has to know the mappings - quite often this is in a file “/etc/ethers”

Since this information is generally in a file, RARP servers are generally implemented as **user processes** (because a kernel process should **not** do file I/O!)

- Unlike ARP responses which are generally part of the TCP/IP implementation (often part of the kernel).
- How does the process get the packets - since they aren’t IP and won’t come across a socket?
  - BSD Packet filters
  - SVR4 Data Link Provider Interface (DLPI)
  - SUN’s Network Interface Tap (NIT)
  - Interestingly in the appendix to RFC 903 an alternative to having data link level access was to have two IOCTLs, one that would "sleep until there is a translation to be done, then pass the request out to the user process"; the other means: "enter this translation into the kernel table"
- RARP requests are sent as hardware level broadcasts - therefore are **not** forwarded across routers:
  - multiple servers per segment - so in case one is down; the first response is used
  - having the router answer

IK2555, SPRING 2014 SLIDE 33

R. Finlayson, T. Mann, J. C. Mogul, and M. Theimer, ‘A Reverse Address Resolution Protocol’, *Internet Request for Comments*, vol. RFC 903 (INTERNET STANDARD), Jun. 1984 [Online]. Available: http://www.rfc-editor.org/rfc/rfc903.txt

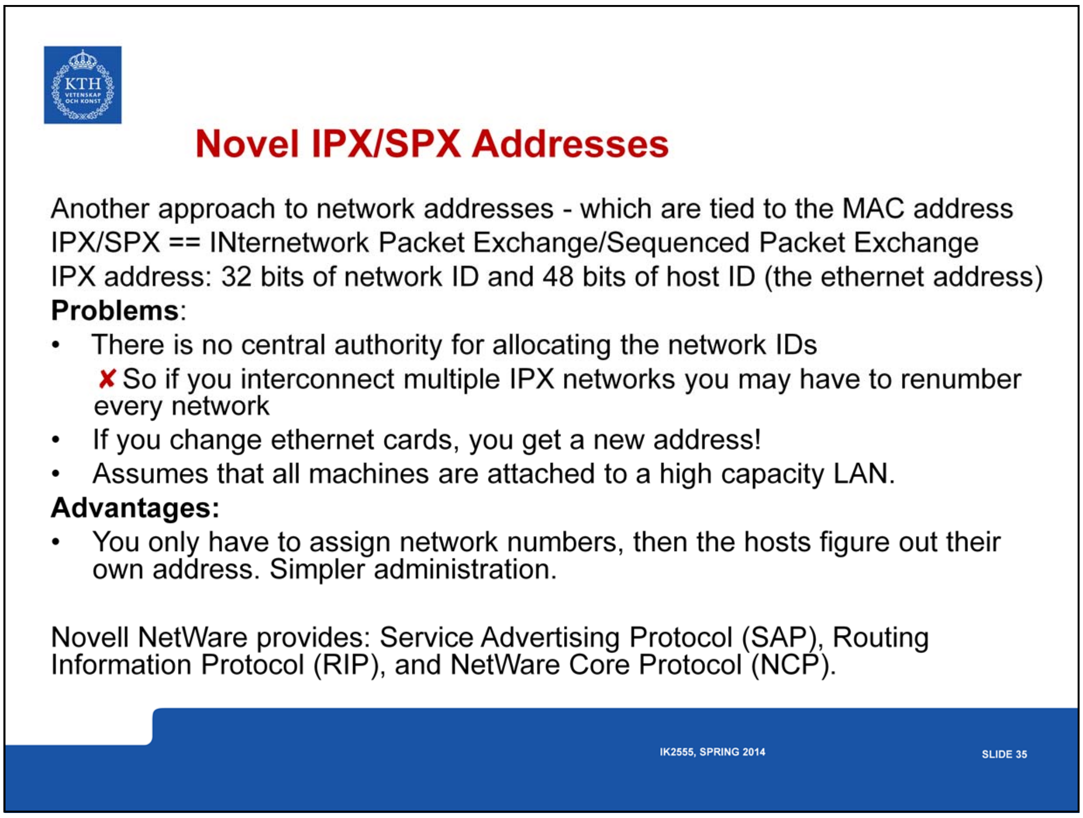


Alternatives to RARP

In a later lecture we will examine:

- BOOTP and DHCP (for both IPv4 and IPv6) and
- autoconfiguration for IPv6.

IK2555, SPRING 2014 SLIDE 34



Novel IPX/SPX Addresses

Another approach to network addresses - which are tied to the MAC address
IPX/SPX == INternetwork Packet Exchange/Sequenced Packet Exchange
IPX address: 32 bits of network ID and 48 bits of host ID (the ethernet address)

Problems:

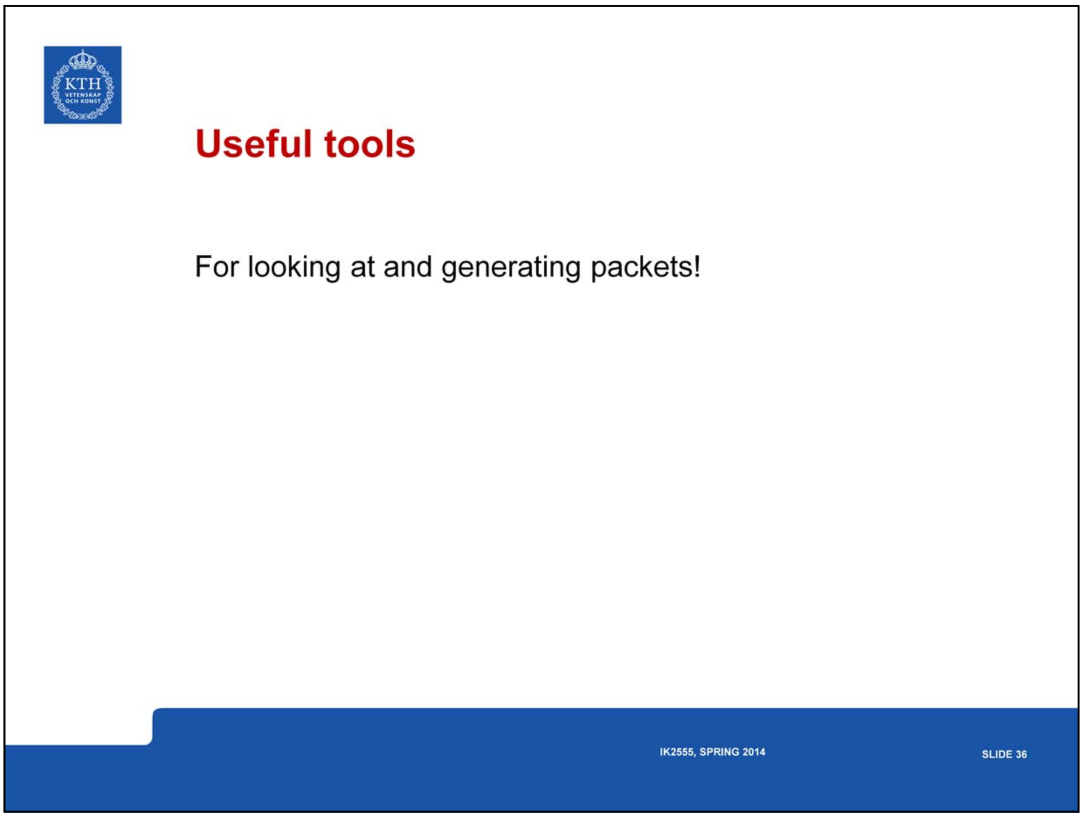
- There is no central authority for allocating the network IDs
  - ✘ So if you interconnect multiple IPX networks you may have to renumber every network
- If you change ethernet cards, you get a new address!
- Assumes that all machines are attached to a high capacity LAN.

Advantages:

- You only have to assign network numbers, then the hosts figure out their own address. Simpler administration.

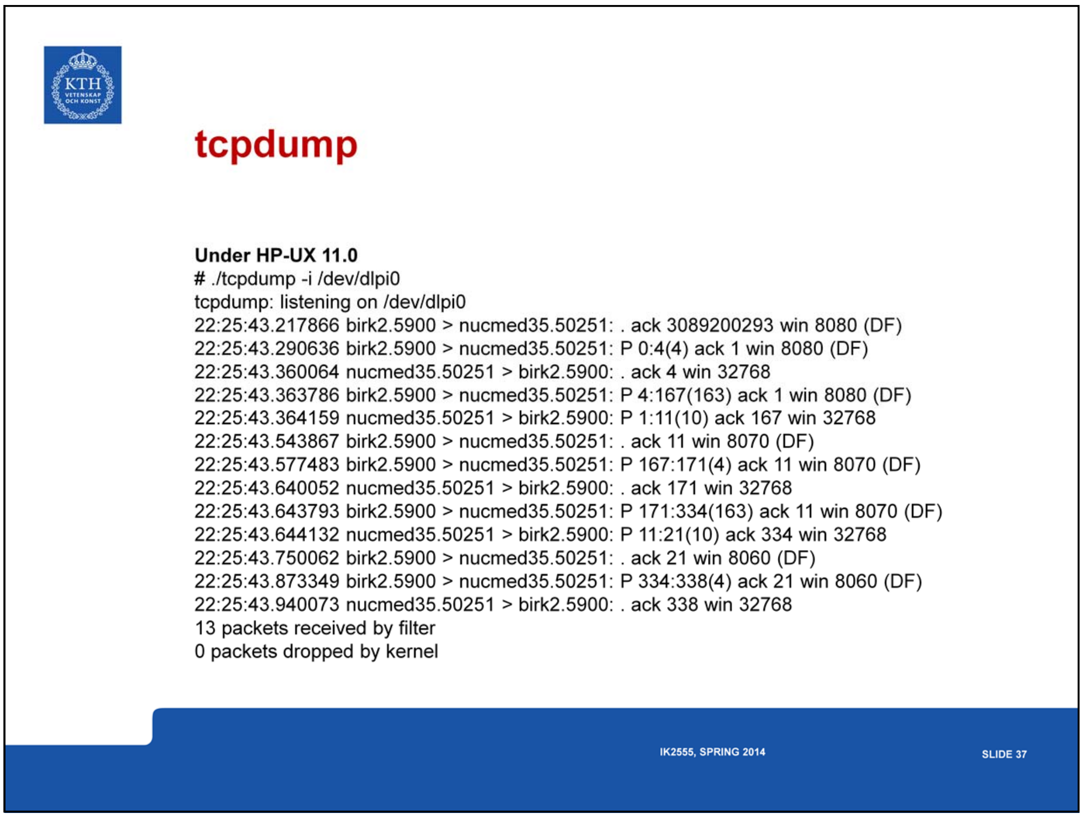
Novell NetWare provides: Service Advertising Protocol (SAP), Routing Information Protocol (RIP), and NetWare Core Protocol (NCP).

IK2555, SPRING 2014		SLIDE 35



Useful tools

For looking at and generating packets!

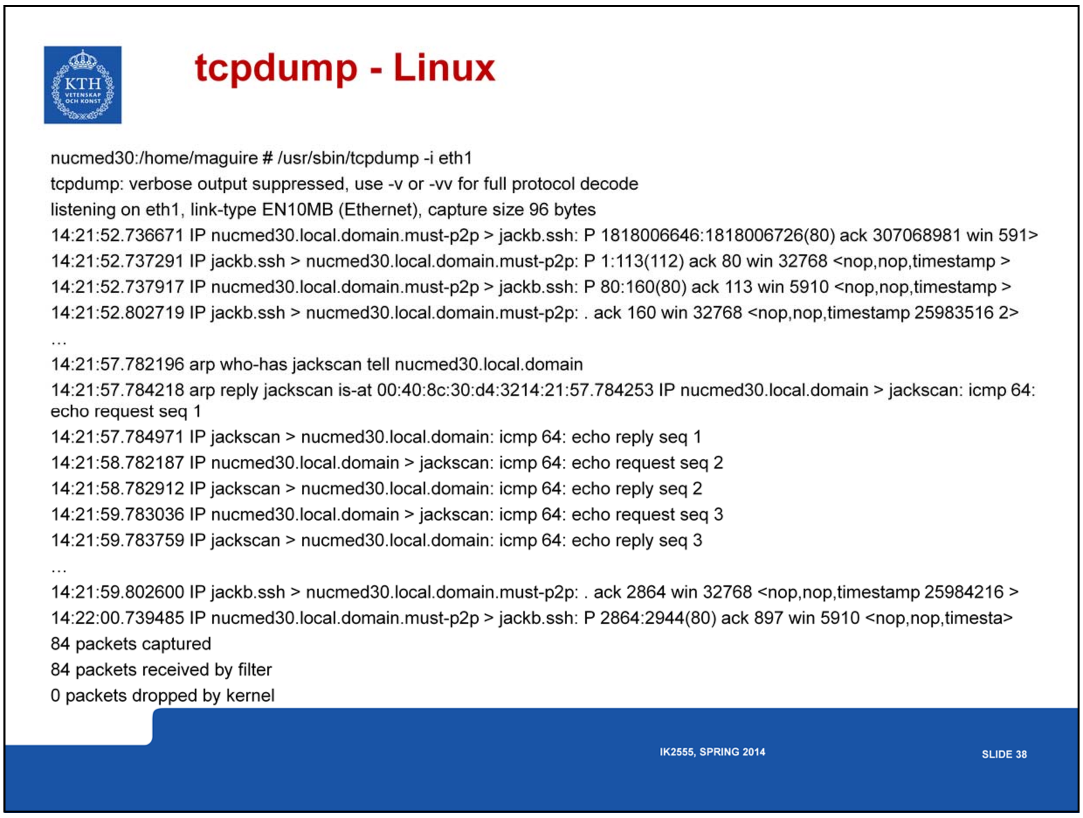


tcpdump

Under HP-UX 11.0

```
# ./tcpdump -i /dev/dlpi0
tcpdump: listening on /dev/dlpi0
22:25:43.217866 birk2.5900 > nucmed35.50251: . ack 3089200293 win 8080 (DF)
22:25:43.290636 birk2.5900 > nucmed35.50251: P 0:4(4) ack 1 win 8080 (DF)
22:25:43.360064 nucmed35.50251 > birk2.5900: . ack 4 win 32768
22:25:43.363786 birk2.5900 > nucmed35.50251: P 4:167(163) ack 1 win 8080 (DF)
22:25:43.364159 nucmed35.50251 > birk2.5900: P 1:11(10) ack 167 win 32768
22:25:43.543867 birk2.5900 > nucmed35.50251: . ack 11 win 8070 (DF)
22:25:43.577483 birk2.5900 > nucmed35.50251: P 167:171(4) ack 11 win 8070 (DF)
22:25:43.640052 nucmed35.50251 > birk2.5900: . ack 171 win 32768
22:25:43.643793 birk2.5900 > nucmed35.50251: P 171:334(163) ack 11 win 8070 (DF)
22:25:43.644132 nucmed35.50251 > birk2.5900: P 11:21(10) ack 334 win 32768
22:25:43.750062 birk2.5900 > nucmed35.50251: . ack 21 win 8060 (DF)
22:25:43.873349 birk2.5900 > nucmed35.50251: P 334:338(4) ack 21 win 8060 (DF)
22:25:43.940073 nucmed35.50251 > birk2.5900: . ack 338 win 32768
13 packets received by filter
0 packets dropped by kernel
```

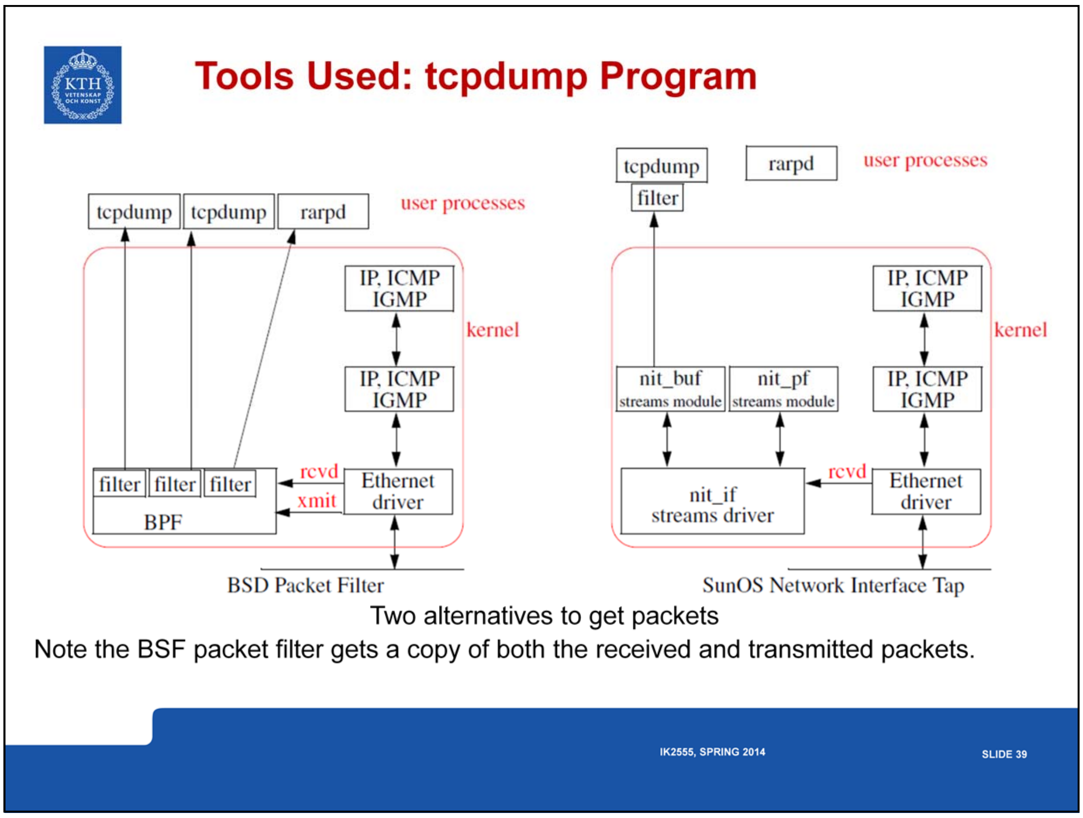
IK2555, SPRING 2014		SLIDE 37



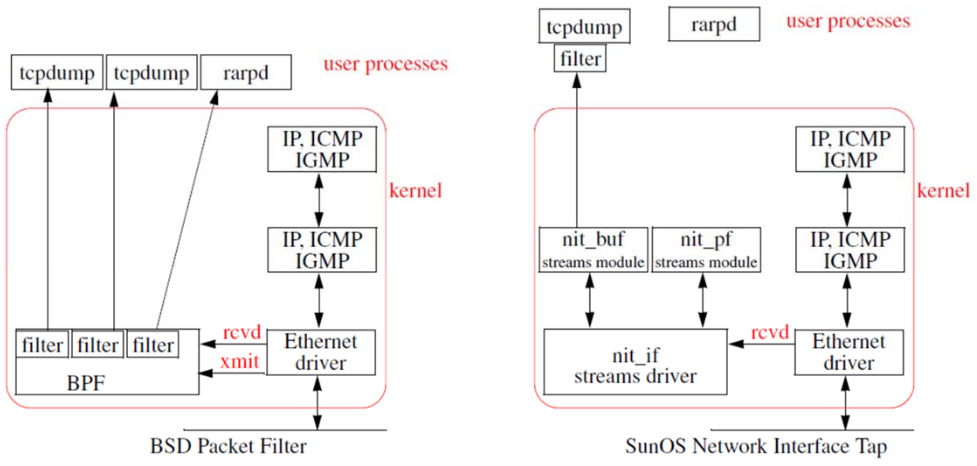
tcpdump - Linux

```
nucmed30:/home/maguire # /usr/sbin/tcpdump -i eth1
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth1, link-type EN10MB (Ethernet), capture size 96 bytes
14:21:52.736671 IP nucmed30.local.domain.must-p2p > jackb.ssh: P 1818006646:1818006726(80) ack 307068981 win 591>
14:21:52.737291 IP jackb.ssh > nucmed30.local.domain.must-p2p: P 1:113(112) ack 80 win 32768 <nop,nop,timestamp >
14:21:52.737917 IP nucmed30.local.domain.must-p2p > jackb.ssh: P 80:160(80) ack 113 win 5910 <nop,nop,timestamp >
14:21:52.802719 IP jackb.ssh > nucmed30.local.domain.must-p2p: . ack 160 win 32768 <nop,nop,timestamp 25983516 2>
…
14:21:57.782196 arp who-has jackscan tell nucmed30.local.domain
14:21:57.784218 arp reply jackscan is-at 00:40:8c:30:d4:3214:21:57.784253 IP nucmed30.local.domain > jackscan: icmp 64:
echo request seq 1
14:21:57.784971 IP jackscan > nucmed30.local.domain: icmp 64: echo reply seq 1
14:21:58.782187 IP nucmed30.local.domain > jackscan: icmp 64: echo request seq 2
14:21:58.782912 IP jackscan > nucmed30.local.domain: icmp 64: echo reply seq 2
14:21:59.783036 IP nucmed30.local.domain > jackscan: icmp 64: echo request seq 3
14:21:59.783759 IP jackscan > nucmed30.local.domain: icmp 64: echo reply seq 3
…
14:21:59.802600 IP jackb.ssh > nucmed30.local.domain.must-p2p: . ack 2864 win 32768 <nop,nop,timestamp 25984216 >
14:22:00.739485 IP nucmed30.local.domain.must-p2p > jackb.ssh: P 2864:2944(80) ack 897 win 5910 <nop,nop,timesta>
84 packets captured
84 packets received by filter
0 packets dropped by kernel
```

IK2555, SPRING 2014 SLIDE 38



Tools Used: tcpdump Program

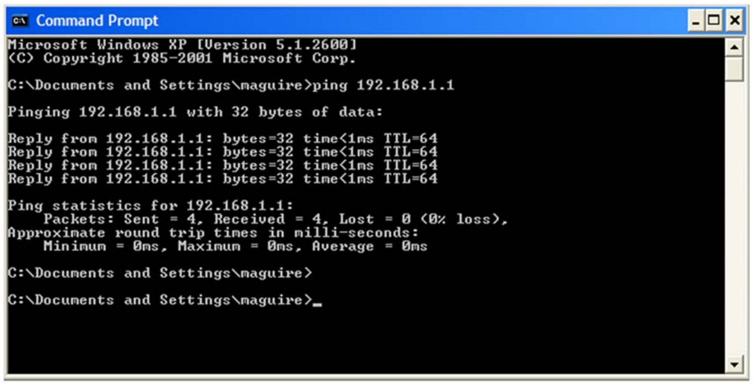


Two alternatives to get packets

Note the BSF packet filter gets a copy of both the received and transmitted packets.

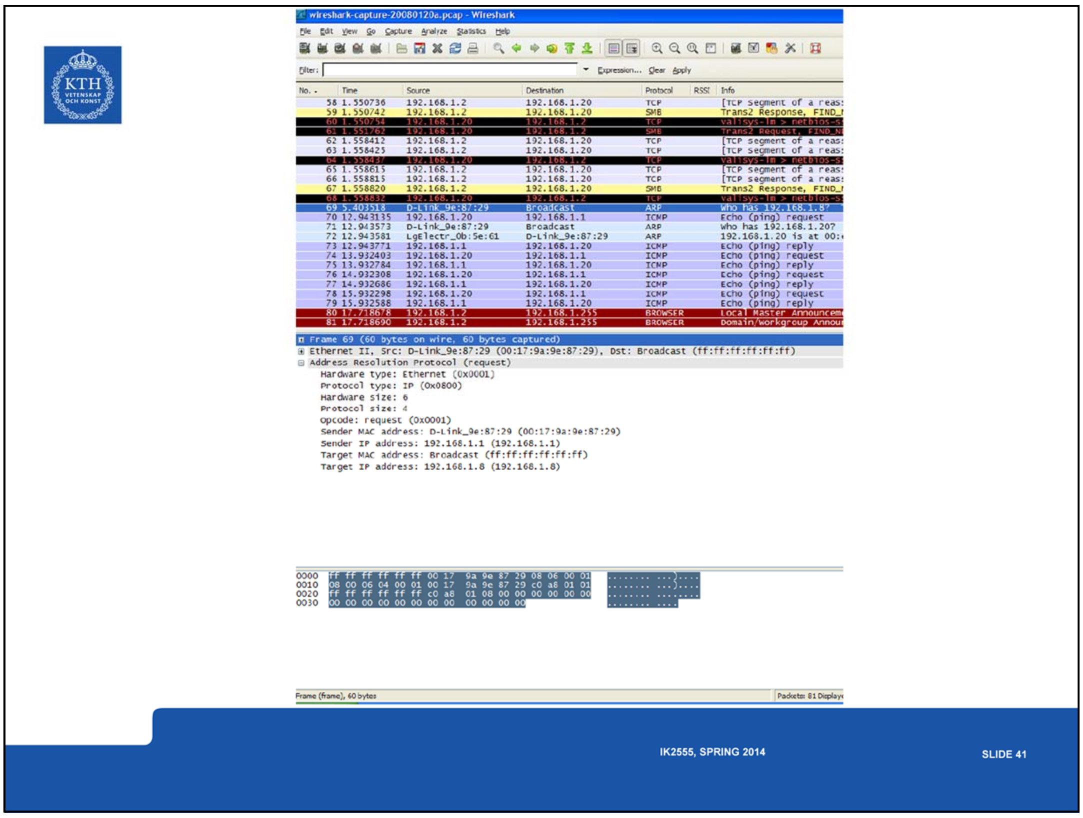
Wireshark (formerly Ethereal)

First we start Wireshark capturing packets, then we ping another machine on the LAN:



```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\maguire>ping 192.168.1.1

Pinging 192.168.1.1 with 32 bytes of data:

Reply from 192.168.1.1: bytes=32 time<1ms TTL=64
Reply from 192.168.1.1: bytes=32 time<1ms TTL=64
Reply from 192.168.1.1: bytes=32 time<1ms TTL=64
Reply from 192.168.1.1: bytes=32 time<1ms TTL=64

Ping statistics for 192.168.1.1:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms

C:\Documents and Settings\maguire>

C:\Documents and Settings\maguire>_
```

Ping another machine on the LAN


wireshark-capture-20080120a.pcap - Wireshark
File Edit View Go Capture Analyze Statistics Help
Filter:
Expression... Clear Apply

```
No. .  Time       Source             Destination        Protocol  RSSI  Info
    58 1.550736   192.168.1.2        192.168.1.20       TCP             [TCP segment of a reas
    59 1.550742   192.168.1.2        192.168.1.20       SMB             Trans2 Response, FIND_
    60 1.550754   192.168.1.20       192.168.1.2        TCP             valisys-lm > netbios-s
    61 1.551762   192.168.1.20       192.168.1.2        SMB             Trans2 Request, FIND_N
    62 1.558412   192.168.1.2        192.168.1.20       TCP             [TCP segment of a reas
    63 1.558425   192.168.1.2        192.168.1.20       TCP             [TCP segment of a reas
    64 1.558437   192.168.1.20       192.168.1.2        TCP             valisys-lm > netbios-s
    65 1.558615   192.168.1.2        192.168.1.20       TCP             [TCP segment of a reas
    66 1.558815   192.168.1.2        192.168.1.20       TCP             [TCP segment of a reas
    67 1.558820   192.168.1.2        192.168.1.20       SMB             Trans2 Response, FIND_
    68 1.558832   192.168.1.20       192.168.1.2        TCP             valisys-lm > netbios-s
    69 5.403518   D-Link_9e:87:29    Broadcast          ARP             Who has 192.168.1.8?
    70 12.943135  192.168.1.20       192.168.1.1        ICMP            Echo (ping) request
    71 12.943573  D-Link_9e:87:29    Broadcast          ARP             Who has 192.168.1.20?
    72 12.943581  LgElectr_0b:5e:61  D-Link_9e:87:29    ARP             192.168.1.20 is at 00:
    73 12.943771  192.168.1.1        192.168.1.20       ICMP            Echo (ping) reply
    74 13.932403  192.168.1.20       192.168.1.1        ICMP            Echo (ping) request
    75 13.932784  192.168.1.1        192.168.1.20       ICMP            Echo (ping) reply
    76 14.932308  192.168.1.20       192.168.1.1        ICMP            Echo (ping) request
    77 14.932686  192.168.1.1        192.168.1.20       ICMP            Echo (ping) reply
    78 15.932298  192.168.1.20       192.168.1.1        ICMP            Echo (ping) request
    79 15.932588  192.168.1.1        192.168.1.20       ICMP            Echo (ping) reply
    80 17.718678  192.168.1.2        192.168.1.255      BROWSER         Local Master Announcem
    81 17.718690  192.168.1.2        192.168.1.255      BROWSER         Domain/workgroup Annou
```

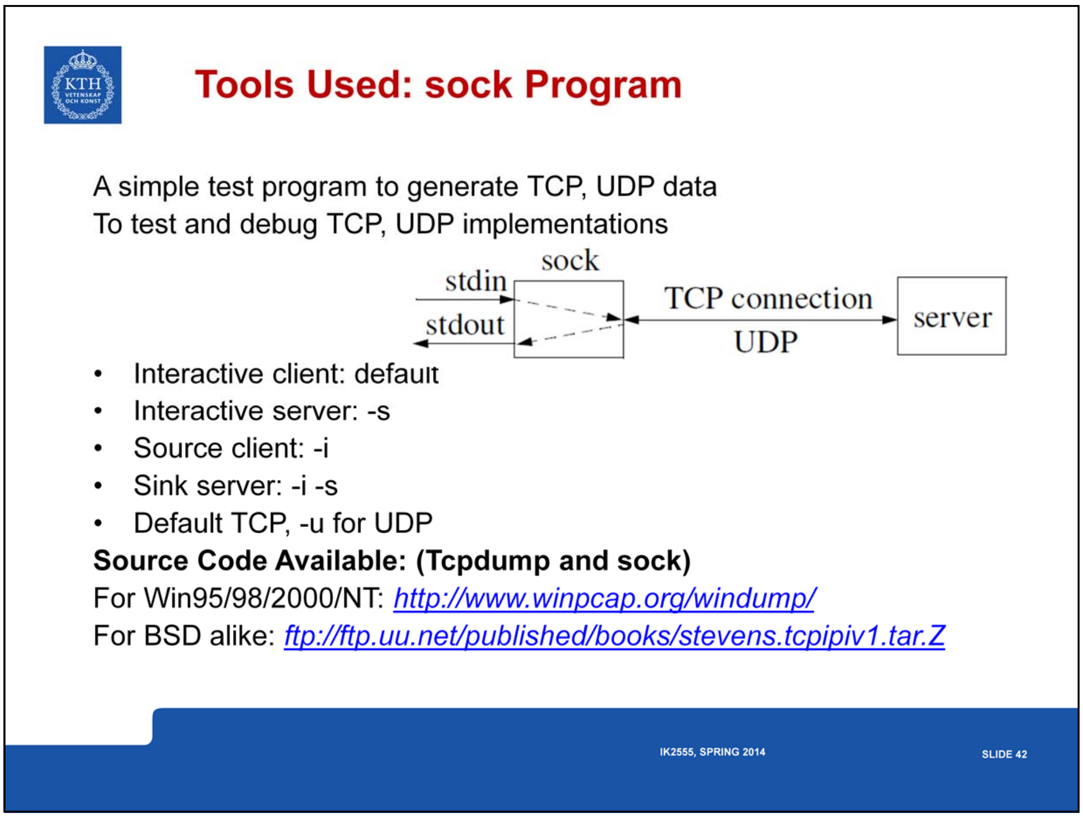
```
⊞ Frame 69 (60 bytes on wire, 60 bytes captured)
⊞ Ethernet II, Src: D-Link_9e:87:29 (00:17:9a:9e:87:29), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
⊟ Address Resolution Protocol (request)
    Hardware type: Ethernet (0x0001)
    Protocol type: IP (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: request (0x0001)
    Sender MAC address: D-Link_9e:87:29 (00:17:9a:9e:87:29)
    Sender IP address: 192.168.1.1 (192.168.1.1)
    Target MAC address: Broadcast (ff:ff:ff:ff:ff:ff)
    Target IP address: 192.168.1.8 (192.168.1.8)
```

```
0000  ff ff ff ff ff ff 00 17  9a 9e 87 29 08 06 00 01   ........ ...)....
0010  08 00 06 04 00 01 00 17  9a 9e 87 29 c0 a8 01 01   ........ ...)....
0020  ff ff ff ff ff ff c0 a8  01 08 00 00 00 00 00 00   ........ ........
0030  00 00 00 00 00 00 00 00  00 00 00 00               ........ ....
```

Frame (frame), 60 bytes | Packets: 81 Display

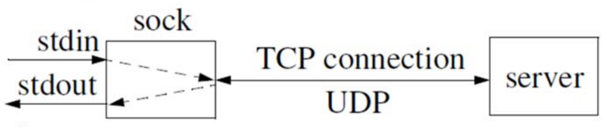
IK2555, SPRING 2014

SLIDE 41



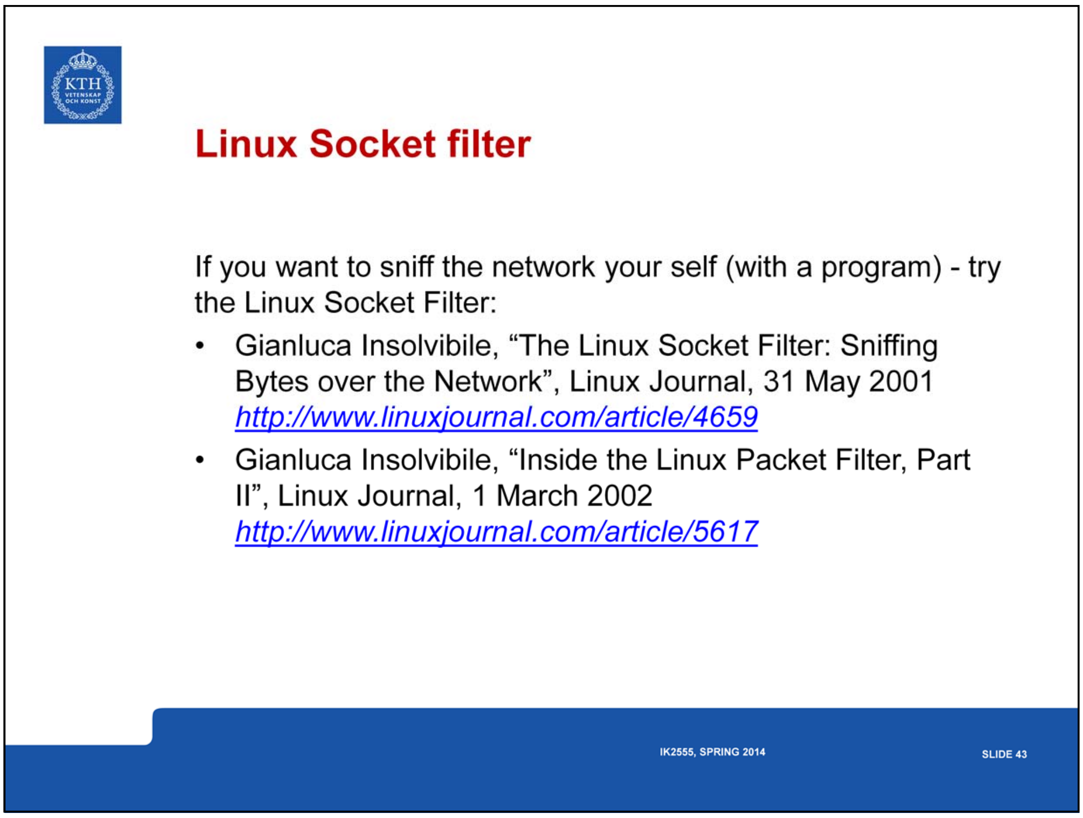
Tools Used: sock Program

A simple test program to generate TCP, UDP data
To test and debug TCP, UDP implementations



- Interactive client: default
- Interactive server: -s
- Source client: -i
- Sink server: -i -s
- Default TCP, -u for UDP

Source Code Available: (Tcpdump and sock)
For Win95/98/2000/NT: *http://www.winpcap.org/windump/*
For BSD alike: *ftp://ftp.uu.net/published/books/stevens.tcpipiv1.tar.Z*

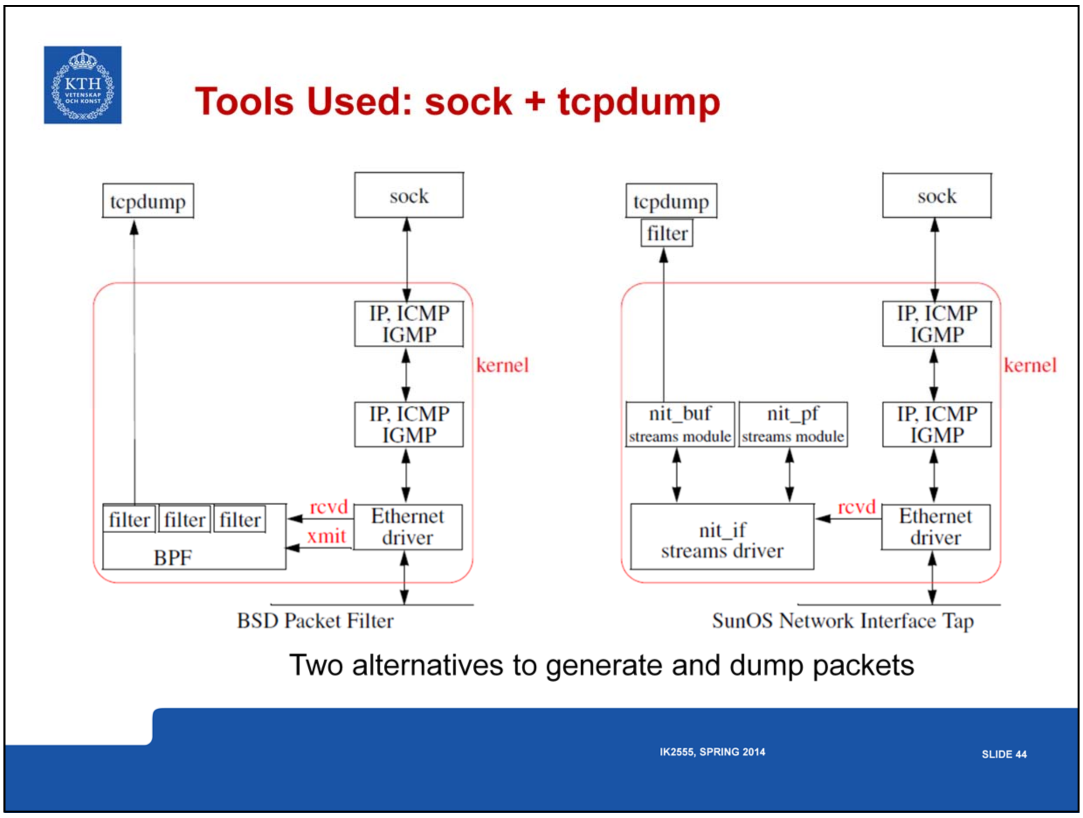


Linux Socket filter

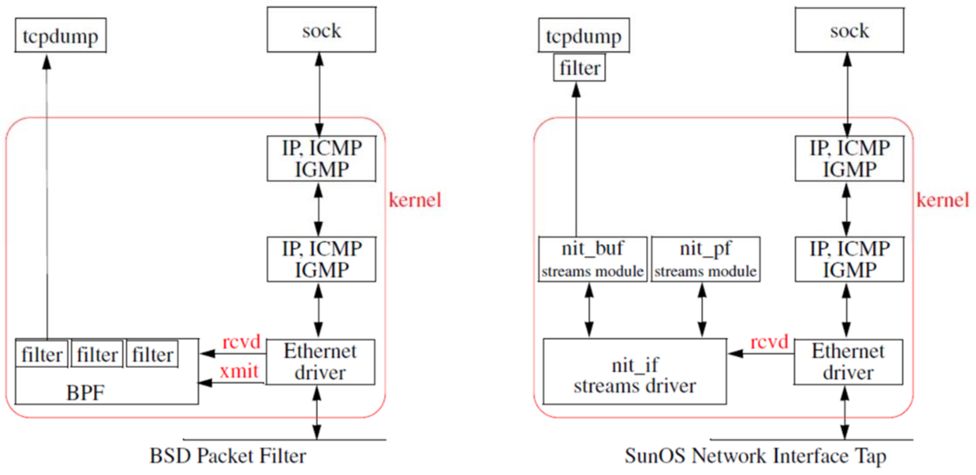
If you want to sniff the network your self (with a program) - try the Linux Socket Filter:

- Gianluca Insolvibile, “The Linux Socket Filter: Sniffing Bytes over the Network”, Linux Journal, 31 May 2001 *http://www.linuxjournal.com/article/4659*
- Gianluca Insolvibile, “Inside the Linux Packet Filter, Part II”, Linux Journal, 1 March 2002 *http://www.linuxjournal.com/article/5617*

IK2555, SPRING 2014 SLIDE 43

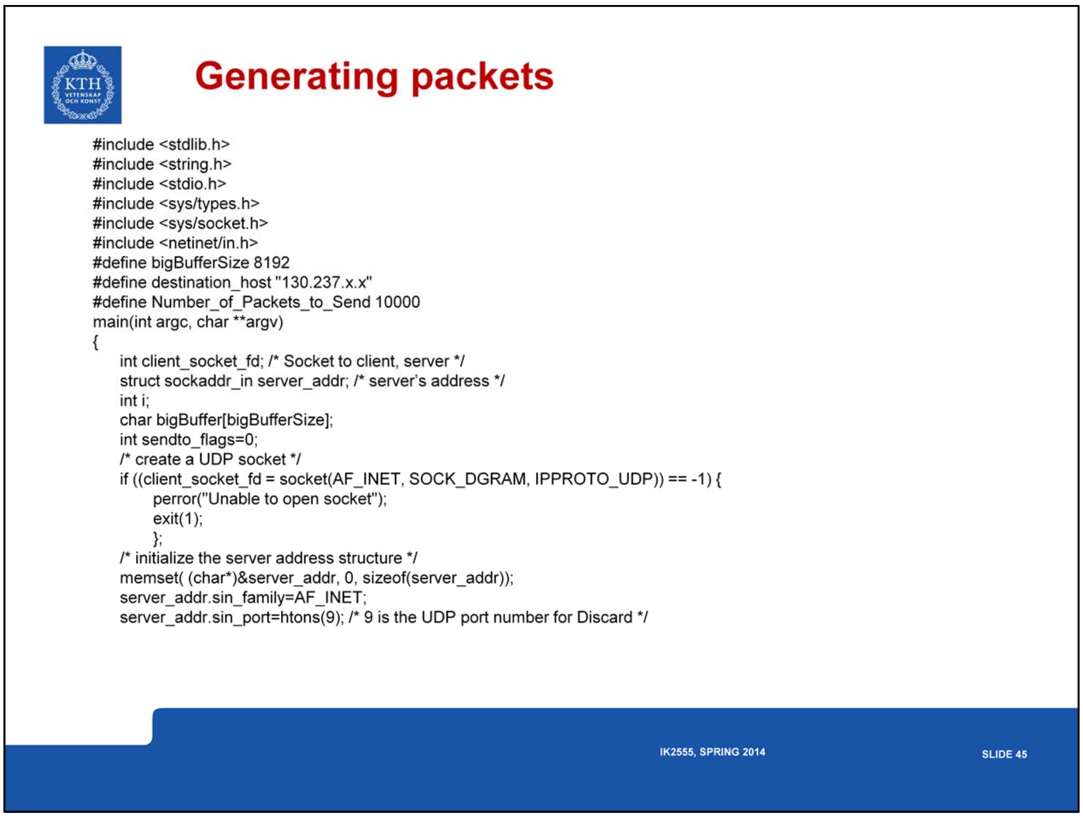


Tools Used: sock + tcpdump



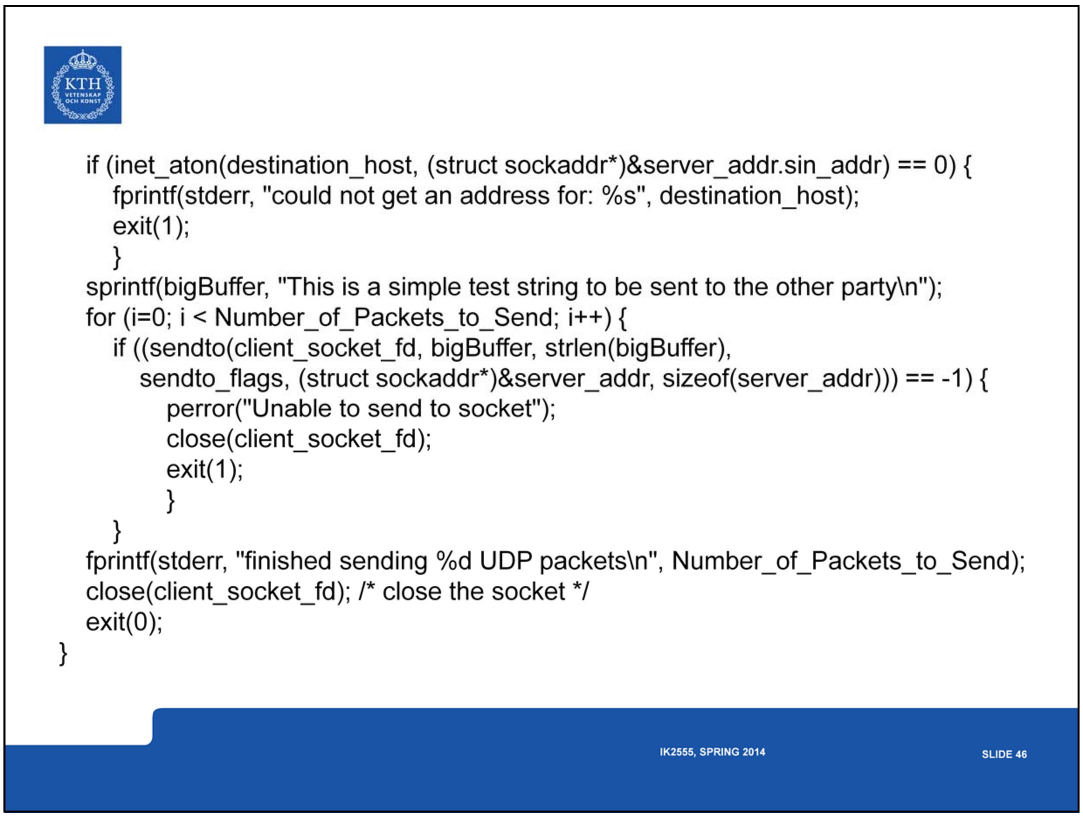
Two alternatives to generate and dump packets

IK2555, SPRING 2014 SLIDE 44



Generating packets

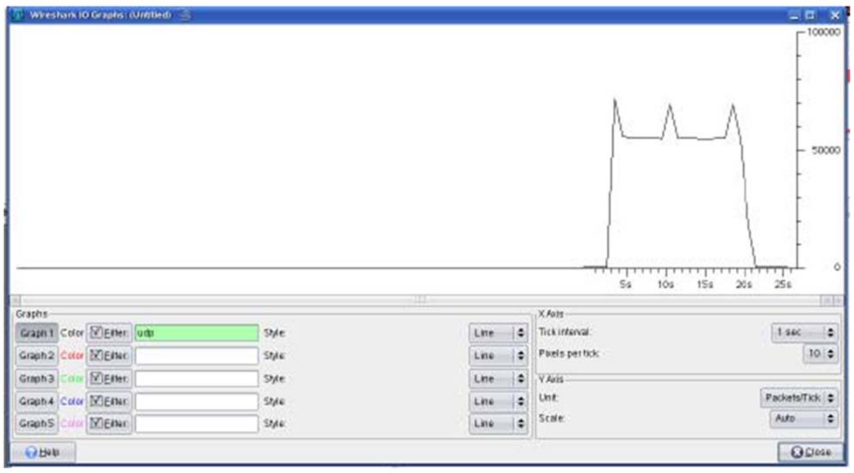
```
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#define bigBufferSize 8192
#define destination_host "130.237.x.x"
#define Number_of_Packets_to_Send 10000
main(int argc, char **argv)
{
    int client_socket_fd; /* Socket to client, server */
    struct sockaddr_in server_addr; /* server's address */
    int i;
    char bigBuffer[bigBufferSize];
    int sendto_flags=0;
    /* create a UDP socket */
    if ((client_socket_fd = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)) == -1) {
        perror("Unable to open socket");
        exit(1);
        };
    /* initialize the server address structure */
    memset( (char*)&server_addr, 0, sizeof(server_addr));
    server_addr.sin_family=AF_INET;
    server_addr.sin_port=htons(9); /* 9 is the UDP port number for Discard */
```



```
    if (inet_aton(destination_host, (struct sockaddr*)&server_addr.sin_addr) == 0) {
        fprintf(stderr, "could not get an address for: %s", destination_host);
        exit(1);
        }
    sprintf(bigBuffer, "This is a simple test string to be sent to the other party\n");
    for (i=0; i < Number_of_Packets_to_Send; i++) {
        if ((sendto(client_socket_fd, bigBuffer, strlen(bigBuffer),
            sendto_flags, (struct sockaddr*)&server_addr, sizeof(server_addr))) == -1) {
                perror("Unable to send to socket");
                close(client_socket_fd);
                exit(1);
                }
        }
    fprintf(stderr, "finished sending %d UDP packets\n", Number_of_Packets_to_Send);
    close(client_socket_fd); /* close the socket */
    exit(0);
}
```

IK2555, SPRING 2014 SLIDE 46

Wireshark’s IO Graph functionality



Graphs

Graph 1 Color ☑ Filter: udp Style: Line
Graph 2 Color ☑ Filter: Style: Line
Graph 3 Color ☑ Filter: Style: Line
Graph 4 Color ☑ Filter: Style: Line
Graph 5 Color ☑ Filter: Style: Line

X Axis

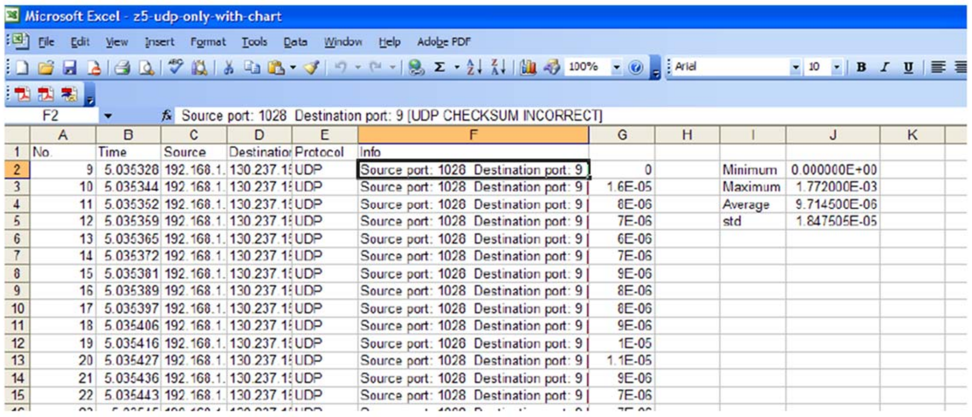
Tick interval: 1 sec
Pixels per tick: 10

Y Axis

Unit: Packets/Tick
Scale: Auto

Plot showing number of packets per second

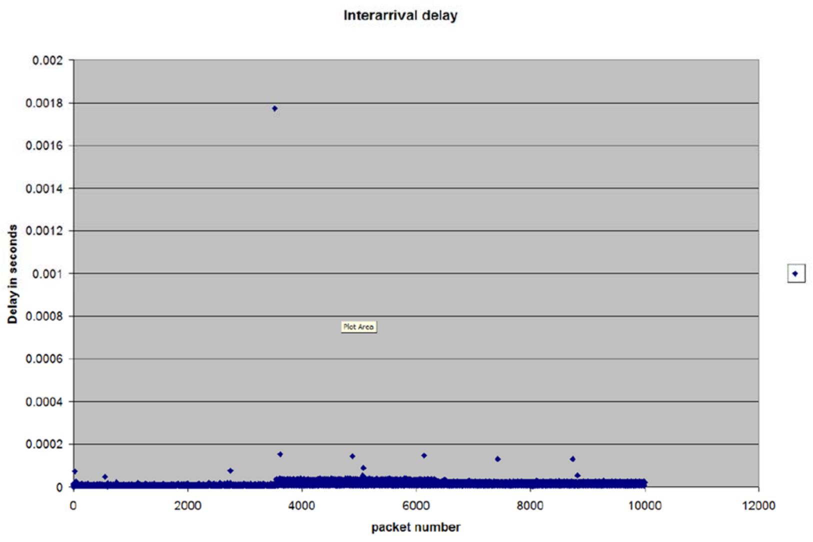
Some statistics on this packet trace


Microsoft Excel - z5-udp-only-with-chart
File Edit View Insert Format Tools Data Window Help Adobe PDF
100%
Arial
10

F2 Source port: 1028 Destination port: 9 [UDP CHECKSUM INCORRECT]

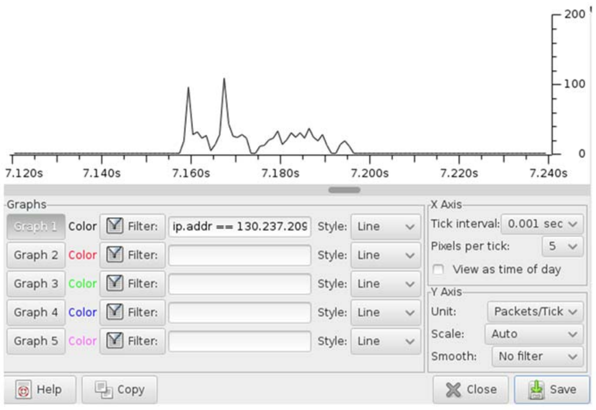
	A	B	C	D	E	F	G	H	I	J	K
1	No.	Time	Source	Destinatior	Protocol	Info					
2	9	5.035328	192.168.1.	130.237.1	UDP	Source port: 1028 Destination port: 9	0		Minimum	0.000000E+00	
3	10	5.035344	192.168.1.	130.237.1	UDP	Source port: 1028 Destination port: 9	1.6E-05		Maximum	1.772000E-03	
4	11	5.035352	192.168.1.	130.237.1	UDP	Source port: 1028 Destination port: 9	8E-06		Average	9.714500E-06	
5	12	5.035359	192.168.1.	130.237.1	UDP	Source port: 1028 Destination port: 9	7E-06		std	1.847505E-05	
6	13	5.035365	192.168.1.	130.237.1	UDP	Source port: 1028 Destination port: 9	6E-06				
7	14	5.035372	192.168.1.	130.237.1	UDP	Source port: 1028 Destination port: 9	7E-06				
8	15	5.035381	192.168.1.	130.237.1	UDP	Source port: 1028 Destination port: 9	9E-06				
9	16	5.035389	192.168.1.	130.237.1	UDP	Source port: 1028 Destination port: 9	8E-06				
10	17	5.035397	192.168.1.	130.237.1	UDP	Source port: 1028 Destination port: 9	8E-06				
11	18	5.035406	192.168.1.	130.237.1	UDP	Source port: 1028 Destination port: 9	9E-06				
12	19	5.035416	192.168.1.	130.237.1	UDP	Source port: 1028 Destination port: 9	1E-05				
13	20	5.035427	192.168.1.	130.237.1	UDP	Source port: 1028 Destination port: 9	1.1E-05				
14	21	5.035436	192.168.1.	130.237.1	UDP	Source port: 1028 Destination port: 9	9E-06				
15	22	5.035443	192.168.1.	130.237.1	UDP	Source port: 1028 Destination port: 9	7E-06				

Interarrival delay and variance



Note the packets which have very high delays

IO Graph functionality (sending a new burst – 2014.03.24)



Graphs

Graph 1	Color	✓ Filter:	ip.addr == 130.237.209	Style:	Line
Graph 2	Color	✓ Filter:		Style:	Line
Graph 3	Color	✓ Filter:		Style:	Line
Graph 4	Color	✓ Filter:		Style:	Line
Graph 5	Color	✓ Filter:		Style:	Line

X Axis

Tick interval: 0.001 sec

Pixels per tick: 5

☐ View as time of day

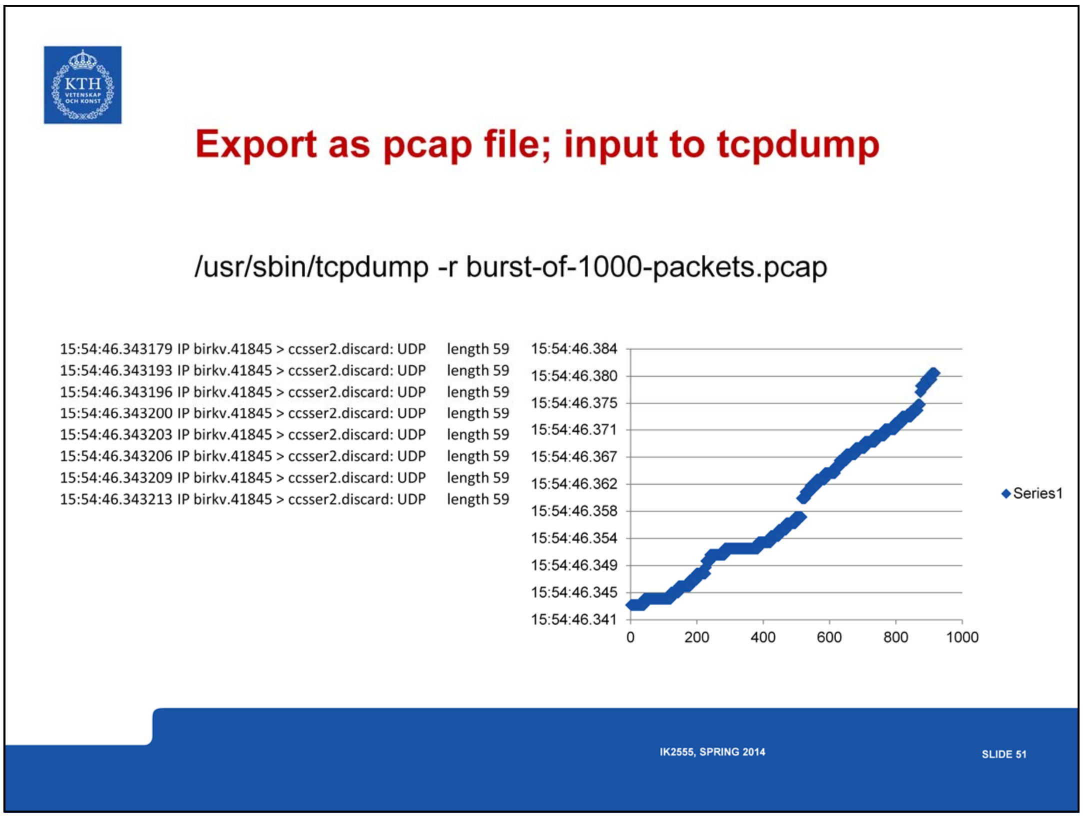
Y Axis

Unit: Packets/Tick

Scale: Auto

Smooth: No filter

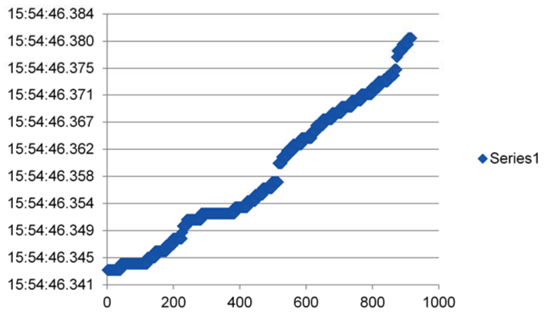
Burst of 1000 packets from my home machine to my office machine



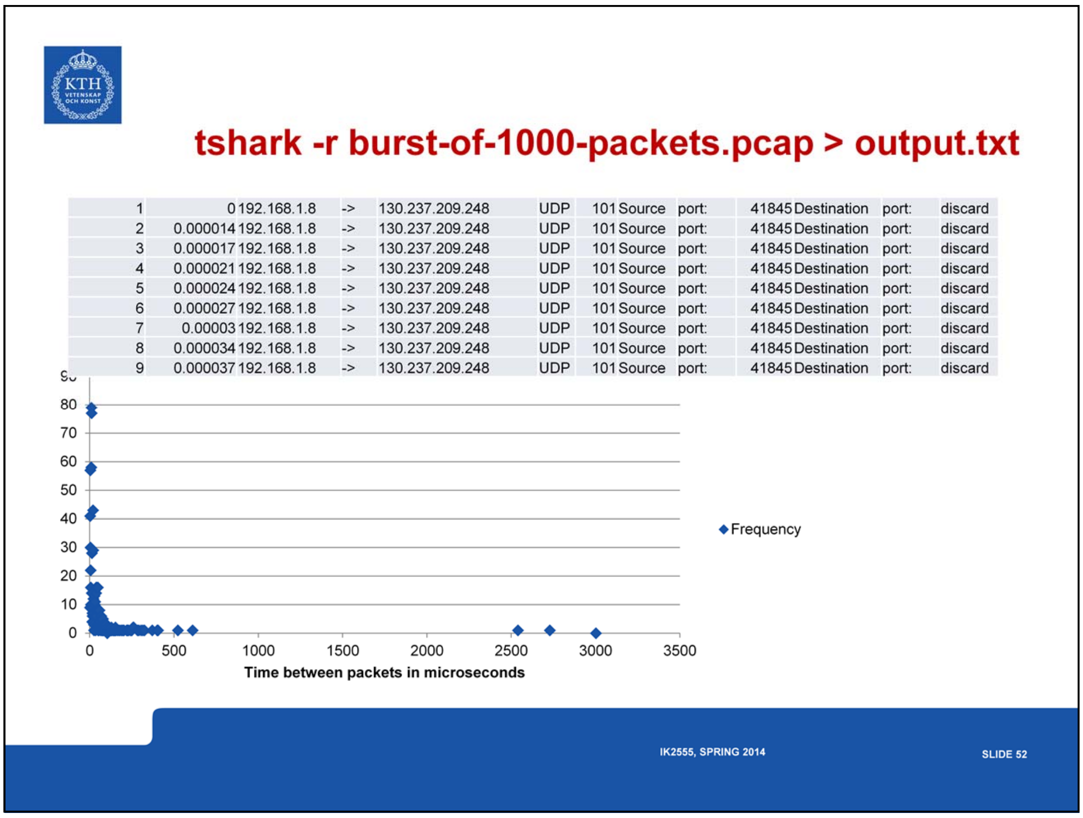
Export as pcap file; input to tcpdump

```
/usr/sbin/tcpdump -r burst-of-1000-packets.pcap
```

```
15:54:46.343179 IP birkv.41845 > ccsser2.discard: UDP    length 59
15:54:46.343193 IP birkv.41845 > ccsser2.discard: UDP    length 59
15:54:46.343196 IP birkv.41845 > ccsser2.discard: UDP    length 59
15:54:46.343200 IP birkv.41845 > ccsser2.discard: UDP    length 59
15:54:46.343203 IP birkv.41845 > ccsser2.discard: UDP    length 59
15:54:46.343206 IP birkv.41845 > ccsser2.discard: UDP    length 59
15:54:46.343209 IP birkv.41845 > ccsser2.discard: UDP    length 59
15:54:46.343213 IP birkv.41845 > ccsser2.discard: UDP    length 59
```

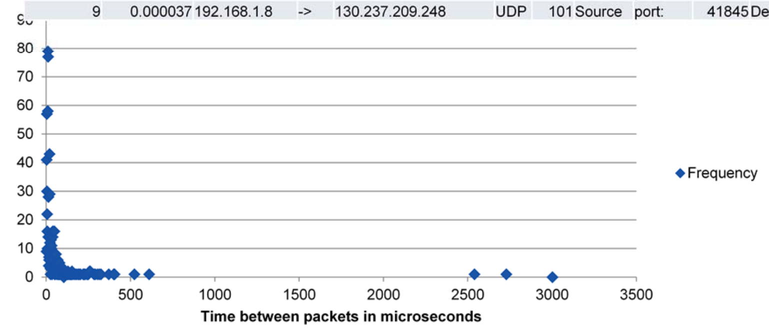


IK2555, SPRING 2014 SLIDE 51



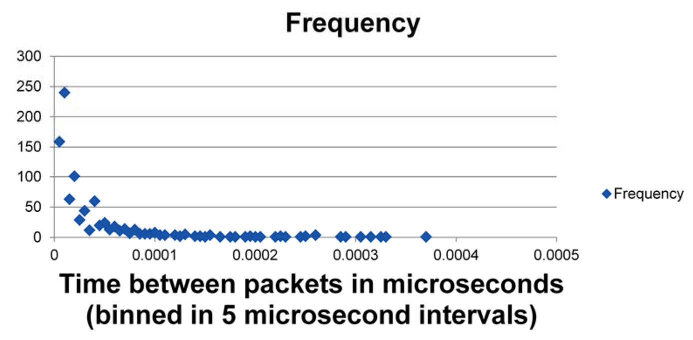
tshark -r burst-of-1000-packets.pcap > output.txt

```
1        0 192.168.1.8  ->  130.237.209.248  UDP  101 Source  port:  41845 Destination  port:  discard
2 0.000014 192.168.1.8  ->  130.237.209.248  UDP  101 Source  port:  41845 Destination  port:  discard
3 0.000017 192.168.1.8  ->  130.237.209.248  UDP  101 Source  port:  41845 Destination  port:  discard
4 0.000021 192.168.1.8  ->  130.237.209.248  UDP  101 Source  port:  41845 Destination  port:  discard
5 0.000024 192.168.1.8  ->  130.237.209.248  UDP  101 Source  port:  41845 Destination  port:  discard
6 0.000027 192.168.1.8  ->  130.237.209.248  UDP  101 Source  port:  41845 Destination  port:  discard
7  0.00003 192.168.1.8  ->  130.237.209.248  UDP  101 Source  port:  41845 Destination  port:  discard
8 0.000034 192.168.1.8  ->  130.237.209.248  UDP  101 Source  port:  41845 Destination  port:  discard
9 0.000037 192.168.1.8  ->  130.237.209.248  UDP  101 Source  port:  41845 Destination  port:  discard
```

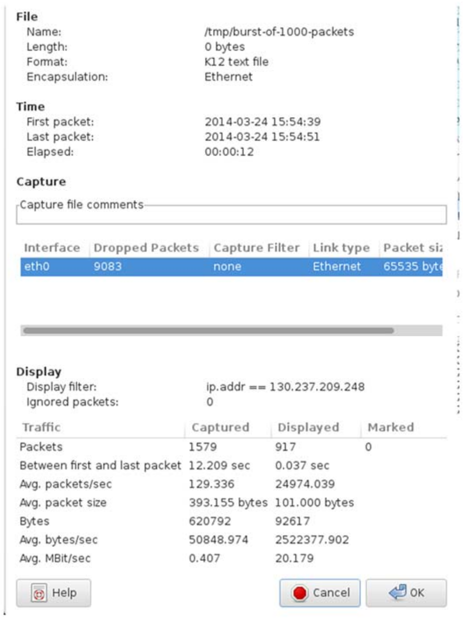


IK2555, SPRING 2014 SLIDE 52

A closer look at the histogram



Summary of burst



File

Name:	/tmp/burst-of-1000-packets
Length:	0 bytes
Format:	K12 text file
Encapsulation:	Ethernet

Time

First packet:	2014-03-24 15:54:39
Last packet:	2014-03-24 15:54:51
Elapsed:	00:00:12

Capture

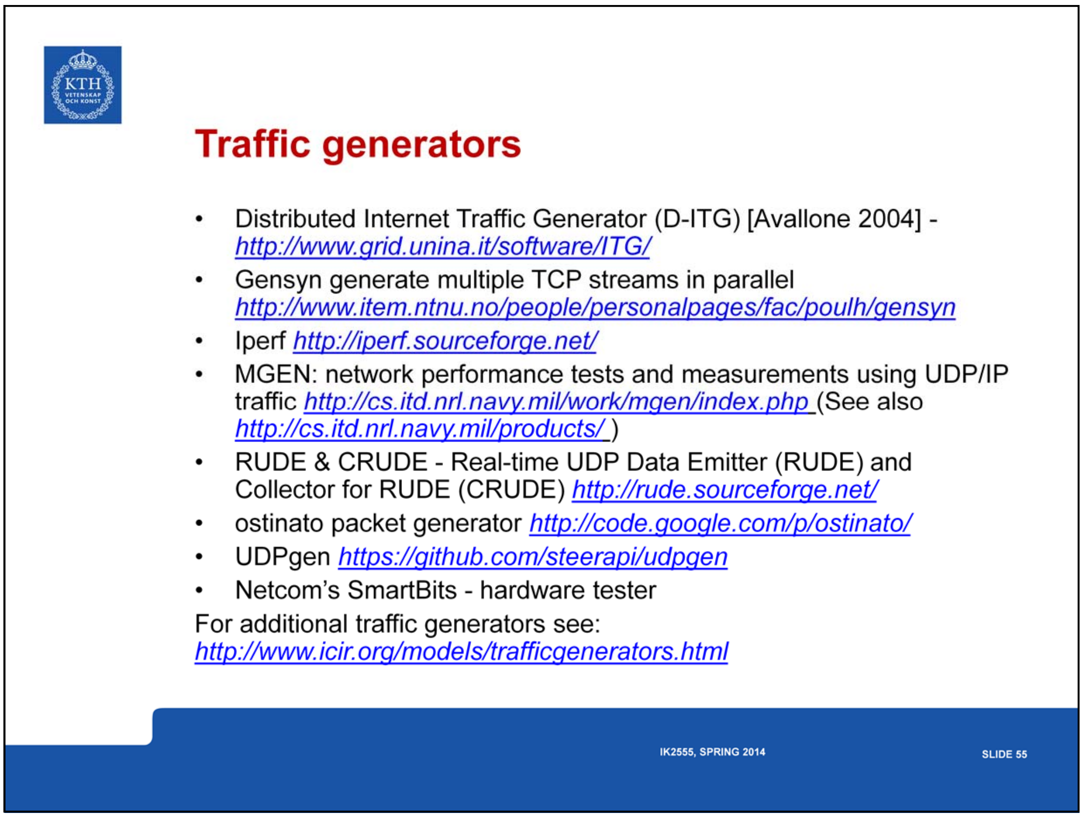
Capture file comments

Interface	Dropped Packets	Capture Filter	Link type	Packet siz
eth0	9083	none	Ethernet	65535 byte

Display

Display filter:	ip.addr == 130.237.209.248
Ignored packets:	0

Traffic	Captured	Displayed	Marked
Packets	1579	917	0
Between first and last packet	12.209 sec	0.037 sec	
Avg. packets/sec	129.336	24974.039	
Avg. packet size	393.155 bytes	101.000 bytes	
Bytes	620792	92617	
Avg. bytes/sec	50848.974	2522377.902	
Avg. MBit/sec	0.407	20.179	



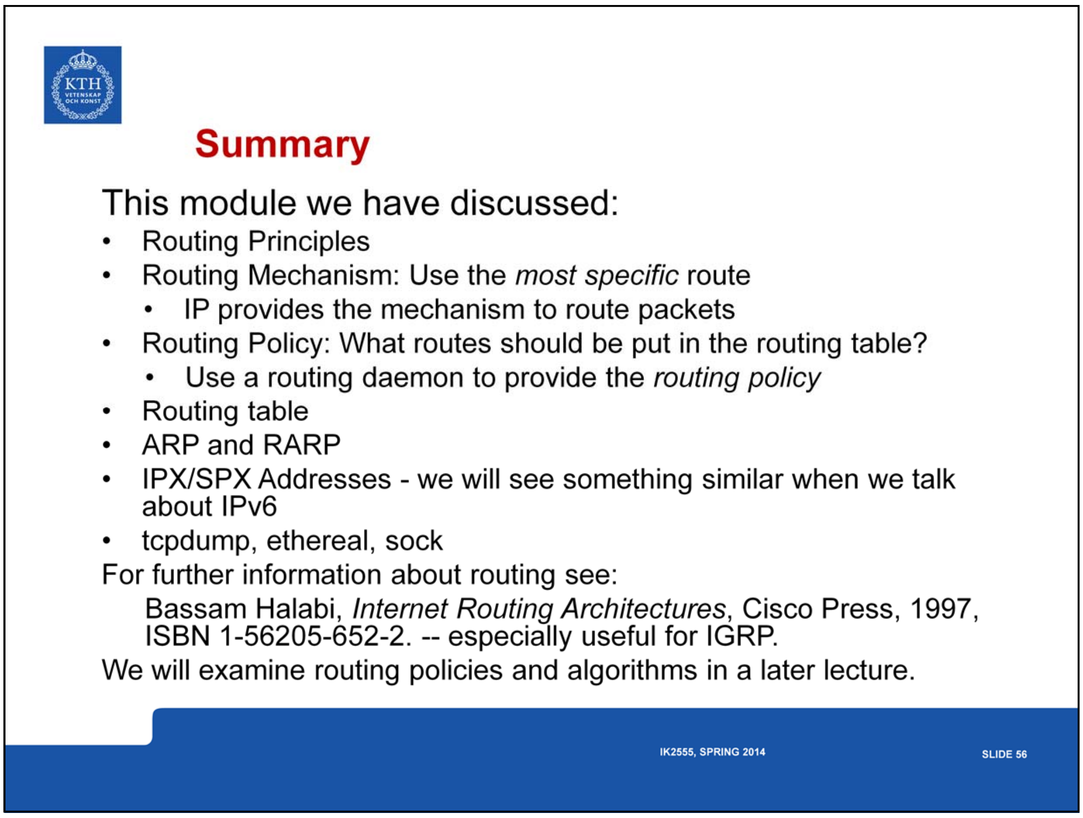
Traffic generators

- Distributed Internet Traffic Generator (D-ITG) [Avallone 2004] - *http://www.grid.unina.it/software/ITG/*
- Gensyn generate multiple TCP streams in parallel *http://www.item.ntnu.no/people/personalpages/fac/poulh/gensyn*
- Iperf *http://iperf.sourceforge.net/*
- MGEN: network performance tests and measurements using UDP/IP traffic *http://cs.itd.nrl.navy.mil/work/mgen/index.php* (See also *http://cs.itd.nrl.navy.mil/products/*)
- RUDE & CRUDE - Real-time UDP Data Emitter (RUDE) and Collector for RUDE (CRUDE) *http://rude.sourceforge.net/*
- ostinato packet generator *http://code.google.com/p/ostinato/*
- UDPgen *https://github.com/steerapi/udpgen*
- Netcom's SmartBits - hardware tester

For additional traffic generators see: *http://www.icir.org/models/trafficgenerators.html*

IK2555, SPRING 2014 SLIDE 55

Stefano Avallone, Antonio Pescapé, and Giorgio Ventre, "Analysis and experimentation of Internet Traffic Generator", International Conference on Next Generation Teletraffic and Wired/Wireless Advanced Networking (NEW2AN'04), February 02-06, 2004 *http://www.grid.unina.it/software/ITG/D-ITGpubblications/New2an-ITG.pdf*



Summary

This module we have discussed:

- Routing Principles
- Routing Mechanism: Use the *most specific* route
  - IP provides the mechanism to route packets
- Routing Policy: What routes should be put in the routing table?
  - Use a routing daemon to provide the *routing policy*
- Routing table
- ARP and RARP
- IPX/SPX Addresses - we will see something similar when we talk about IPv6
- tcpdump, ethereal, sock

For further information about routing see:

Bassam Halabi, *Internet Routing Architectures*, Cisco Press, 1997, ISBN 1-56205-652-2. -- especially useful for IGRP.

We will examine routing policies and algorithms in a later lecture.

IK2555, SPRING 2014 SLIDE 56

¿Questions?

IK1550, SPRING 2014

SLIDE 57