Lecture 3 Overview

- Paper reviews
- Packet Forwarding
- IP Addressing
  - Subnetting/CIDR
- Data-plane processing
  - NAT
  - Middle boxes
Paper review tips

- Demonstrate you *understood* the paper
  - Don’t just give short synopses
  - Analogies/examples are good ways to demonstrate thought

- Understand the context of the paper
  - Where/when was it published? By whom?

- Think outside of the box
  - We selected “good” papers
  - Shortcomings are likely to lie outside of their stated scope

- We’re looking for implications they didn’t state
  - What impact did/could this paper have beyond the goals listed specifically in the paper
WHAT ARE THE MAIN CONTRIBUTIONS?
This paper provides a guidance on at which level certain functions should be implemented, with respect to data communication. Many of these functions are necessary to ensure that the application on top runs correctly, and in fact are crucial components to the Internet. For example, an online airline's booking system may not reserve the same seat for two persons, and would need some mechanisms to ensure that this could not happen (what would the service do if two customers sent a "reserve seat 42A" at the same time?). Also, when customers are sending their credit card information over a network, they would like to know that it is impossible (that is, very difficult) to extract it anywhere between the sender and receiver.

Such functionality (or network guarantees) may or may not be common to many applications. It can thus be tempting to add this functionality at multiple levels of the layered communication model, but that can affect performance badly. This paper suggests us to think twice of where to put certain functionality, using the end-to-end argument. The lower layers might implement some of the functionality, but only for performance reasons. With respect to reliable connections, striving for a negligible error rate below the application level is probably not a good idea. Something can go wrong on each side outside this layer, so caution must be taken anyway. This leads to redundant functionality, which constrains performance.

WHAT ARE THE MAIN SHORTCOMINGS?
The paper does not, however, give us a complete solution of how communication should be done - and where to put certain functions. If all desired functionality needs to be implemented at the application level, you would probably see similar code in all applications. From a programmers point of view, you would like an easy interface "that just works". This is something we see with TCP sockets: It gives the programmer a reliable connection to be used in the application layer, with delivery guarantees and data integrity. Using the end-to-end argument as a guideline can help us build fast and reliable data communication, but it is important to know it's limitations.

WHAT ARE THE IMPLICATIONS OF THIS PAPER?
Moving functionality away from the center of a network and out to its hosts/end systems can have huge benefits. Routers and switches that constitutes the network center can be made much simpler, thus allowing operation at higher speeds. They may not be able to provide guarantees like "all packets sent from router A to router B will be received successfully", but because of the simplicity of the underneath design it can be more efficient. If a router's buffer is full and it receives a packet, it can simply be dropped because it is the application's responsibility to send and receive the packet safely. The switches and routers do not even need to keep any state when operating (i.e. they can be stateless); they mostly do dummywork as "move this incoming packet to this outgoing link". This simple "network core" can be seen on the Internet today; the Internet provides no guarantees by itself, it just happen to work very well most of the time.
What are the main contributions?

This paper explored the "End-to-End" argument or more simply put, argues that it is the application, both server and client (the "Ends"), that should provide the needed functionality (i.e., encryption, reliability, ordering, message duplication handling) instead of the communication subsystems making assumptions about what functionality the application might need and preemptively providing them. This is due to the fact that the application knows best what it is trying to accomplish and as such should provide for itself. Also, by providing functionality at the lower levels you force anything built upon that level to use that functionality and thus pay for it even when the functionality is not needed or even worse is actually detrimental to the task at hand, such as with VOIP where packet loss is preferable over reliable transmission. Lastly, there are cases where only the application can determine correctness or completion, such as when writing to a file remotely, where the receiving end will need to generate the reply only once it has finished writing and NOT when it has first gotten the message to write.

In short, the paper is warning developers against the temptation to create a one size fits all communication layer since there are many different problems which are unique enough to need many different solutions and thus the simpler and smaller a layer below is the more flexible it will be to build upon.

What are the main shortcomings?

One of the biggest shortcomings of this paper is that I'm damn lazy and really not interested in adding reliability and congestion control and in-order transmission to my application. I want it, but I don't want to work for it. Furthermore, if I attempt to write it myself, I'm most certainly going to get it wrong, and next thing you know, I'm flooding the network with retries and ruining everybody's day. Par for the course, I know.

But in all seriousness, by not providing a communication layer that provides some of the more common connection models, you force application developers to re-invent the wheel, which in the real world cost dollar dollar dollar bills, yo.

Furthermore, you actually have more to gain by having time tested protocols that know how to properly behave in corner cases such as congestion and failures. It's like when you are driving down the freeway. It's nice knowing everyone on the right hand side of the road is moving the same direction as you are; it's one less thing to worry about and makes things more predictable.

Even more furthermore, if you hand a programmer and hammer, he's going to start looking at all his problems as nails. With a protocol such as TCP, the programmer starts to model his problem around the tools provided to get the performance he/she needs so long as TCP provides the bulk of the functionality required and "good enough" performance. This is good, because if we know what a hammer looks like we can start planning optimizations around a hammer's needs and hopefully give a boost to all applications that work well with hammers.

Now I know it can be argued that protocols such as TCP are acting on top of a simpler protocol IP that handles the most basic communication, as called for in the paper, and that we could then start thinking of protocols such as TCP as a library an application could use and thus start thinking of TCP as part of the application or "Ends". While this is true to a degree, the
Motivation

- Goal: efficiently deliver packets between arbitrary hosts in the network
- Key concerns: scalability, performance, robustness
- Key techniques:
  - Hierarchical design
    » Learn enough to do part of the job, hand task off to someone else
    » e.g., one technique for inter-domain routing, another for intra-domain
  - Soft state
    » Learn all the information you need to perform routing
    » State info not for correctness, just performance optimizations
Forwarding Options

- Source routing (Myrinet)
  - Packet carries path
- Table of global addresses (IP)
  - Stateless routers
- Table of virtual circuits (ATM)
  - Small headers, small tables

► How do hosts/switches learn optimal network routes?
  - Given packet header, how to determine forwarding port
  - Topic for Thursday…
Comparison

<table>
<thead>
<tr>
<th></th>
<th>Source routing</th>
<th>Global addresses</th>
<th>Virtual circuits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header size</td>
<td>worst</td>
<td>OK ~ large addr</td>
<td>best</td>
</tr>
<tr>
<td>Router table size</td>
<td>none</td>
<td># of hosts (prefixes)</td>
<td># of circuits</td>
</tr>
<tr>
<td>Forward overhead</td>
<td>best</td>
<td>Prefix matching</td>
<td>Pretty good</td>
</tr>
<tr>
<td>Setup overhead</td>
<td>none</td>
<td>none</td>
<td>High</td>
</tr>
<tr>
<td>Error recovery</td>
<td>Tell all hosts</td>
<td>Tell all routers</td>
<td>Tear down circuit and reroute</td>
</tr>
</tbody>
</table>
Packet Forwarding

- Control plane computes a forwarding table
  - Maps destination address(es) to an output link

- Handling an incoming packet
  - Match: destination address
  - Action: direct the packet to the chosen output link

- Switching fabric
  - Directs packet from input link to output link
Switch: Match on MAC

- MAC addresses are location independent
  - Assigned by the vendor of the interface card
  - Cannot be aggregated across hosts in the LAN

Implemented using a hash table or a content addressable memory.
IP Routers: Match on IP Prefix

- IP addresses grouped into common subnets
  - Allocated by ICANN, regional registries, ISPs, and within individual organizations
  - Variable-length prefix identified by a mask length

Prefixes may be nested.
Routers identify the **longest matching** prefix.
Address Resolution Protocol

- IP forwarding tables: one entry per network not host
  - Thus, routes designed to get packets to proper network
  - Network needs to take over from there to get to proper host

- Address resolution protocol (ARP) translates IP addresses to link-level addresses (e.g., Ethernet addr)
  - Broadcast request over network for IP ➔ link-level mapping
  - Maintain local cache (with timeout)
Broadcast: Anyone know the Ethernet address for 152.3.140.5?

Reply: Yes, I’m at 08-00-2b-18-bc-65 (152.3.145.240)

Packet arrives for host on Same physical network

Ethernet

(152.3.145.240) (152.3.140.5)
Internet: Hierarchical Routing

- Internet composed of many autonomous systems (AS’s)
  - Correspond to administrative domains

- Each AS can choose its own routing algorithm
  - Routing Information Protocol (RIP) used originally
    - Part of BSD distribution, distance vector
  - Open Shortest Path First (OSPF) currently most popular
    - Link state protocol w/authentication, basic load balancing

- Border Gateway Protocol (BGP) for routing between AS’s
  - Default: shortest number of AS’s in path
  - Sys admins can express policy control
    - Use AS $x$ in preference to AS $y$
IP: The Internet Protocol

- Service mode: *best effort*
  - No guarantees about reliable, in-order, or error-free delivery
  - Enables IP to “run over anything”

<table>
<thead>
<tr>
<th>Version</th>
<th>HLen</th>
<th>TOS</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Identi</td>
<td>Flags</td>
<td>Offset</td>
<td></td>
</tr>
<tr>
<td>TTL</td>
<td>Protocol</td>
<td>Checksum</td>
<td></td>
</tr>
<tr>
<td>SourceAddr</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DestinationAddr</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Options (variable)</td>
<td>Pad (variable)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data…</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Problem: networks have different maximum transmission units (MTUs)
- Ethernet: 1500 bytes, FDDI: 4500 bytes, etc.
- Communicating hosts may be on networks w/similar MTUs
  But smaller MTU somewhere in the middle of the network

To maintain uniform host-to-host communication, IP must fragment and then reassemble packets on end-to-end basis

Hosts pick MTU of local network

Fragmentation takes place at routers that forward packets along links with smaller MTU
- Input on 1500-byte MTU link, output on 500-byte MTU link
IP Address Allocation

- Originally (classfull addr), 4 address classes
  - “A”: 0 | 7 bit network | 24 bit host (1M each)
  - “B”: 10 | 14 bit network | 16 bit host (64K)
  - “C”: 110 | 21 bit network | 8 bit host (255)
  - “D”: 1110 | 28 bit multicast group #

- Assign net # centrally, host # locally
  - E.g., assign UCSD a class B address
IP Address Issues

- We can run out
  - 4B IP addresses

- We’ll run out faster if sparsely allocated
  - Rigid structure causes internal fragmenting
  - E.g., assign a class C address to site with 2 computers
    Waste 99% of assigned address space

- Need address aggregation to keep tables small
  - 2 million class C networks
  - Entry per network in IP forwarding tables
    Scalability?
Efficient IP Address Allocation

- **Subnets**
  - Split net addresses between multiple sites

- **Supernets**
  - Assign adjacent net addresses to same organization
  - Classless routing (CIDR)
    - Combine routing table entries whenever all nodes with same prefix share same hop

- **Hardware support for fast prefix lookup**
CIDR

- Classless Interdomain Routing (CIDR)
  - Balances entries in forwarding tables with need to efficiently distribute IP address space

- Example: site that requires 16 class-C IP addresses
  - Use 16 contiguous class C addrs, e.g., 192.4.16-192.4.31
  - Top 20 bits are identical
  - Have created something between a class B and class C addr “Classless”

- Need routing protocols to recognize CIDR
  - BGP-4 sends updates in form <length,value> (e.g., where length is 20 for above example).
Network Address Translation

- Allows multiple machines to be assigned same IPV4 addr
- NAT separates internal from external hosts
  - Hosts only need internally unique address
- NAT translates each packet
  - Multiplex small set of externally unique addresses among active connections
  - Internal IP -> dynamically allocated ext. IP
- What if NAT crashes?
- Interaction with security?
Network Address Translation

CSE 222A – Lecture 3: Packet Forwarding
Mapping Addresses and Ports

- Remap IP addresses and TCP/UDP port numbers
  - **Addresses**: between end-host and NAT addresses
  - **Port numbers**: to ensure each connection is unique

- Create table entries as packets arrive
  - **Src 10.0.0.1, SPort 1024, Dest 1.2.3.4, DPort 80**
    » Map to **Src 138.76.29.7, Sport 1024, Dest 1.2.3.4, Dport 80**
  - **Src 10.0.0.2, SPort 1024, Dest 1.2.3.4, DPort 80**
    » Map to **Src 138.76.29.7, Sport 1025, Dest 1.2.3.4, Dport 80**

- Challenges
  - When to remove the entries
  - Running services behind a NAT
  - What if both ends of a connection are behind NATs
A Generic Switch Fabric

CSE 222A – Lecture 3: Packet Forwarding
Buffering

- **Drop-tail FIFO queue**
  - Packets served in the order they arrive
  - ... and dropped if queue is full

- **Random Early Detection (RED)**
  - When the buffer is nearly full
  - ... drop or mark some packets to signal congestion

- **Multiple classes of traffic**
  - Separate FIFO queue for each flow or traffic class
  - ... with a link scheduler to arbitrate between them
Link Scheduling

- Strict priority
  - Assign an explicit rank to the queues
  - ... and serve the highest-priority backlogged queue

- Weighted fair scheduling
  - Interleave packets from different queues
  - ... in proportion to weights

50% red, 25% blue, 25% green
Traffic Shaping

- Force traffic to conform with a profile
  - To avoid congesting downstream resources
  - To enforce a contract with the customer

- Leaky-bucket shaping
  - Can send at rate $r$ and intermittently burst
  - Parameters: token rate $r$ and bucket depth $d$

A leaky-bucket shaper for each flow or traffic class
Traffic Classification/Marking

- Mark a packet to influence handling downstream
  - Early Congestion Notification (ECN) flag
  - Type-of-Service (ToS) bits

- Ways to set the ToS bits
  - End host sets the bits based on the application
    » But, then the network must trust (or bill!) the end host
  - Network sets the bits based on traffic classes
    » But, then the network needs to know how to classify packets

- Identifying traffic classes
  - Packet classification based on the “five tuple”
  - Rate limits, with separate mark for “out of profile” traffic
Generalized Data Plane

- Streaming algorithms that act on packets
  - Matching on some bits, taking a simple action
  - ... at behest of control and management plane

- Wide range of functionality
  - Forwarding
  - Access control
  - Mapping header fields
  - Traffic monitoring
  - Buffering and marking
  - Shaping and scheduling
Middle Boxes

- **Router**
  - Forward on destination IP address
  - Access control on the “five tuple”
  - Link scheduling and marking
  - Monitoring traffic
  - Deep packet inspection

- **Switch**
  - Forward on destination MAC address

- **Firewall**
  - Access control on “five tuple” (and more)

- **NAT**
  - Mapping addresses and port numbers

- **Shaper**
  - Classify packets
  - Shape or schedule

- **Packet sniffer**
  - Monitoring traffic
Programmable Data Plane

- Programmable data plane
  - Arbitrary customized packet-handling functionality
  - Building a new data plane, or extending existing one

- Speed is important
  - Data plane in hardware or in the kernel
  - Streaming algorithms that handle packets as they arrive

- Two open platforms
  - Click: software data plane in user space or the kernel
  - NetFPGA: hardware data plane based on FPGAs

- Lots of ongoing research activity…
OpenFlow

- **Match**
  - Match on a subset of bits in the packet header
  - E.g., key header fields (addresses, port numbers, etc.)
  - Well-suited to capitalize on TCAM hardware

- **Action**
  - Perform a simple action on the matching packet
  - E.g., forward, flood, drop, rewrite, count, etc.

- **Controller**
  - Software that installs rules and reads counts
  - ... and handles packets the switch cannot handle
Future of IP

- Is IP really an indirection layer?
  - Mobility
  - Overlay networks
  - Peer to peer networks
  - Application-layer routing
  - NAT

- Perhaps an IP address becomes a convenient intermediate handle for performing the next lookup?
For Next Class…

- Read P&D Chapter 4

- Read and review Paxson ’97
  - (Changed since the beginning of the term)

- Keep thinking about term project ideas/groups
  - Groups and ideas due next week