Lecture 3:
Packet Forwarding

CSE 222A: Computer Communication Networks
Alex C. Snoeren

Thanks: Mike Freedman & Amin Vahdat
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?

The paper presented end-to-end argument, which is an argument for moving functionality to the application-level rather than implementing it at a low-level. The argument revolves around the belief that an application must perform a complete check of correctness, irrespective of any checks or functionality provided by the underlying system. Therefore, the underlying system should be augmented with new capabilities and properties sparingly, and it should always be possible to circumvent those features. The authors presented examples of limiting or redundant shared functionality. A number of these examples show that functionality required in an application in some cases by nature need to be implemented in the actual application, making similar functionality provided by lower layers redundant and some times even limiting. However, having redundant functionality is not always a bad thing, in some cases it actually improves performance.

What are the main shortcomings?

The disadvantage of end-to-end design principle is that it’s difficult to control resources among competing applications. The paper mainly consider either the performance of the applications or the ease of development as metrics to decide the placement of the functionality, however, I think fair share of resources or control of share portions, are also important aspects for the decision for the placement of the functionality. For example, the end-to-end principle can’t guarantee fair share allocation among users if some of users are malicious or miss-behaving.

The authors explained that the end-to-end principle needs the low level enhancement in the case that the function inherently contains unreliable nature. Even in the case, some application finds the cost of the enhancement is not worth the result, or the enhancement makes worse. This implies that there are potential benefits to inform application requirements to the low level functions by some mechanism. I suppose that this can be future work, although author doesn't mention it.

What are the implications of this paper?

The Internet's evolution and growth has been supported by the use of end-to-end arguments to decide where to place functionality in the overall architecture as it evolved, it is a significant factor in the Internet's scalability which is keeping functionality at the end points reduces costs of scaling the core.

The key point in the argument is the fact that different applications and high level services have different requirements in terms of the infrastructure they need in order to achieve their objectives, and that since these requirements are based on the individual nature of the functions and tasks they perform, lower level mechanisms and services should be built as simple as possible. To be more precise, the functionality provided by these lower level mechanisms should be evaluated weighing the cost of implementation of what often times is redundant functionality, versus performance benefits. The goal is a mechanism that is as simple as possible, and yet improves the performance of the application or applications that will run on top.
WHAT ARE THE MAIN CONTRIBUTIONS?
This paper presents the principal of ‘end-to-end’ design, in which designers should at least consider holding off on implementing a feature at a given layer (in a layered or multi-level system) if higher-level layers will need to reimplement that functionality (or similar functionality) anyway. The paper suggests that implementing a feature at the lowest possible layer can come at an unnecessary cost as many higher layers will either not need that feature at all, or will end up reimplementing it anyway. The main possible exception that they acknowledge is when there’s a needed performance boost.

WHAT ARE THE MAIN SHORTCOMINGS?
It’s easy to complain about a paper’s ideas with thirty years of hindsight, but the paper’s examples and writing doesn’t necessarily defend their idea as well as could be hoped for.

The paper’s primary example is that of a file-transfer application. Claiming that because there are so many opportunities for data corruption, implementing reliable data transmission on the network doesn’t buy much, given that the application will still have to verify the same reliability property to account for issues in other components. The paper suggests that since not all components are written by the application programmer, bugs in those other components must be accounted for.

In practice, since many of these components are so commonly used, they’re very well bug tested. Similarly, by reducing the risk of data corruption in each component to a negligible value, the job of the file transfer application is vastly simplified. Even if the application must check for data integrity, by reducing the areas in which a problem may arise, the check by the application gets easier and easier.

Moreover, the paper fails to meaningfully acknowledge the significantly greater cost and risk of bugs introduced by having to reimplement the wheel at each turn. If each application were in charge of its own encryption, very few communications would be secure. And very few applications would be written, as complexity would increase quickly for non-trivial applications. These fundamental layers are almost guaranteed to do a better job at a task simply by nature of having more stress testing, and on the whole, better coders working on them.
The primary gain provided by the 'end to end argument' is preservation of generality. No specific functionality will be precluded or degraded by choices made under consideration of the 'end to end argument'. However, this will be at the cost of uniformity, consistency, and potentially implementation cost. For example, if there's a specific delivery guarantee that is required by the vast majority of applications; each will need to develop their own instance of the algorithm, at their own cost. While the paper argues that this situation is preferable, because there may be individual wrinkles or optimizations for each application, the result is that the applications may not be interoperable later, and that some implementations may be incorrect. Again, as hardware costs go down and development costs rise, it is more important to try to reduce implementation costs and errors. In the context of communications subsystem (network) design, the 'end to end argument' facilitates development of additional distributed applications, but inhibits interoperability and composition of those applications.

As the authors clearly state, they coined a phrase, but not a concept. It's basically a slightly more concrete version of the KISS principle. With respect to contemporary software development, Agile design has simplicity and incremental immediacy as core principles, which is in line with the 'end to end argument' of not overbuilding for potential future use. So their argument is broadly applicable.

Overall, I feel that the argument presented is more explanatory than prescriptive. I think that given the specific goals and hardware availability at the time of the layered network model's origination, the 'end to end argument' happened to apply to the specific design choices that were made. So, in retrospect, it seems like it was a valid guiding principle. However, I don't think it is universally applicable. A high level programming language itself is a subsystem for general purpose future use, and should therefore fall under the 'end to end argument' for deciding what to include. For example, python and R both have the concept of a 'Data Frame' which is specifically designed to represent spreadsheet-like data where repeated trials or instances of data are contained as values aligned along individual rows, where columns represent attributes. This is supremely useful for working with statistical data. However, the standard 'array' is still accessible for scientific computing, images, or other future extensibility. The 'end to end argument' would be explicitly against languages including things such as 'data frames', but that's clearly not the direction of language development.
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

Packet arrives for host on Same physical network
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></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Ident</th>
<th>Flags</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TTL</th>
<th>Protocol</th>
<th>Checksum</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SourceAddr</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>DestinationAddr</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Options (variable)</th>
<th>Pad (variable)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Data…
Fragmentation and Reassembly

- 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

10.0.0.1 → 138.76.29.7

inside

10.0.0.2

outside

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