Advanced Computer Networks
Lecture 2: Principles
Patrick Stuedi
Spring Semester 2013

This week...

Principles:
• Layering and modularity
• Tunnels and virtual networks
• The Internet Hourglass
• In-band vs. Out-of-band signalling
• The End-to-End argument
• Soft state vs. Hard state
• Postel’s Law (robustness principle)
• Fate-sharing

Layering and Modularity

• Decompose system into layers
  – Each layer only relies on services from lower layer
  – Each layer exports services only to layer above

• Interface between layers defines interaction
  – Hide implementation details

ISO/OSI Reference Model

3 key concepts!
1. Service: Tells what the layer does
2. Interface: Tells the process above how to access the layer
3. Protocol: How the service is performed; the layer’s own business.

Last time

• Course introduction
  1. Principles
  2. Wireless networking and mobility
  3. Datacenter- and high-performance network and virtualization

• Recap:
  – Network performance
  – Utility functions
  – Network naming
**Why layering?**

- Dealing with complex systems
- Explicit structure:
  - Identification of complex system’s various pieces
  - Clear relationship between them
- Eases maintenance, updating of system
  - Change of implementation of layer’s service transparent to rest of system
  - E.g. change in gateway procedure doesn’t affect rest of system

**Layering disadvantages**

- Duplication of functionality in multiple layers
  - E.g., error recovery to retransmit lost data
- Multiple layers may need same information
  - E.g. Max Transfer Size (MTU) for TCP segments
- Performance
  - Cannot exploit some per-link-layer techniques
- Headers can be very large
  - Headers can be much larger than payload
- Layer separation is not clean
  - Performance optimizations

**Hardware reality**

- All-in-one box: IP Routing Switch
  - Was sometimes called a “Brouter” (Bridge + Router)
  - Ethernet, VLANs, IP, etc.
  - IP forwarding, multicast, etc.
  - Routing: RIP, OSPF, BGP,
  - Policy routing
  - Etc. etc.
- Question: where are the layers any more?

**Internet protocol stack (TCP/IP reference model)**

- Application
  - HTTP, SMTP, BitTorrent, ...
- Transport
  - Host-host data transfer
  - TCP, UDP, RTP, ...
- Network
  - Routing of datagrams from source to destination
  - IP, routing protocols, ...
- Link
  - Data transfer between neighbouring elements
  - PPP, Ethernet, WiFi, ...
- Physical
  - Bits “on the wire”
  - UTP, Fiber, wireless, ...

**Layer Violations: sometimes a good idea**

- Expose lower-layer information to higher layers
  - E.g. TCP-over-wireless system
  - Pass loss information up to TCP (congestion vs. corruption)
- Expose higher-layer information to lower layers
  - Firewalls
  - Network Address Translators
  - Transparent proxies

**Tunnels and virtual networks**
Tunnelling:

- Many uses!
- A few examples:
  - Personal routing / NAT avoidance (e.g., HTTP Connect)
  - Traffic aggregation and management
  - Security
  - Mobility

Tunnelling: Why?

- Complicates routing
  - Adding additional "links" to a network
  - Statically routed ⇒ suboptimal (ignores routing protocol)
  - Dynamically routed ⇒ routing protocol doesn’t know it’s a tunnel
  - Encapsulation can lead to routing pathologies
- Complicates management / provisioning
  - Unexpected traffic patterns (loops?)
  - Traffic is now “opaque” to the carrier
- Complicates forwarding (for IP)
  - Packets require “shim” header for encapsulation ⇒ reduced MTU, or fragmentation

Tunnel with care…

Tunnelling: a few examples

<table>
<thead>
<tr>
<th>Delivery Protocol</th>
<th>Network</th>
<th>Transport</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network</td>
<td>Transport</td>
<td>Application</td>
<td></td>
</tr>
<tr>
<td>Application</td>
<td>SOCKS</td>
<td>SSL/TLS</td>
<td></td>
</tr>
<tr>
<td>Transport</td>
<td>IPsec</td>
<td>IP-IP</td>
<td>PPTP</td>
</tr>
<tr>
<td></td>
<td>GRE</td>
<td>IPSec</td>
<td></td>
</tr>
<tr>
<td>Link</td>
<td>L2TP</td>
<td>IP over DNS</td>
<td></td>
</tr>
</tbody>
</table>

Mobility

- Problem:
  - When computer moves, address has to change (why?)
  - Breaks ongoing TCP connections
- Solution: computer has two addresses!
  - Locally acquired one (e.g. WiFi in coffee shop)
  - Semi-permanent (acquired from "home" or provider)

  MobileIP: IP traffic sent to semi-permanent address can be tunneled by provider to the local interface (and vice versa)

Virtual private networks

- Idea: use tunnels as link layers
- ⇒ Can build private IP network over tunnels over public IP network.
- Cloud providers sell VPNs
  - Amazon VPN, Hybrid Cloud
- ISPs and Cloud providers sell VPN services to large customers
  - Router support makes this easy (though complex)
  - Probably pays for the Internet...
- Typically IP over IP tunnels
  - GRE, IP/IP, PPTP, AVIYA...

  VPNs are the face of a more general class of Overlay Networks.
**Overlay Networks**

- **Observation:**
  - Can use IP connections as tunnels for other protocols
    - Including IP
  - If you can establish enough “points of presence”, you can run your own network!
    - Routing protocols, addressing, etc.
- **Examples:**
  - Content distribution networks
  - Application-layer multicast
  - RON (Resilient Overlay Networks)
    - Better than IP, over IP!

**Virtual LANs (VLANS)**

- **Problem:**
  - create multiple networks from a single physical network
- **Why?**
  - Isolation, security, management
  - Rewiring is too expensive / difficult / time-consuming

**Virtual LANs (VLANS)**

- **Observe:**
  - Switches are part of multiple virtual networks
  - Hosts are usually on one network
  - Some links are shared between multiple virtual networks

**Moral (contd):**

- **Layers are useful**
  - How else to talk about protocols?
  - Separation of function is important
- **Layers include encapsulation**
  - New layers can be inserted
  - Layers can be “looped” (tunnelling) at any level
- **Encapsulation can be broken**
  - “Deep packet inspection”, combined routing/switching
  - “Cross layer visibility” (expose underlying information)

**The Internet “Hourglass”**

- **Layering by itself does not solve all problems**
- **Many application layers and link layers have evolved**
  - Can’t have every protocol implemented over every other
The Internet “Hourglass”

- IM
- HTTP
- NTP
- DNS
- TCP
- UDP
- IPv4

The “thin waist”

- Single Network-layer protocol: IP
- Easy to absorb new networks
  - just implement IP above
  ⇒ supports innovation below the layer
- Easy to support new applications anywhere
  - just use a transport protocol over IP
  ⇒ supports innovation above the layer
- Changing the layer itself is very difficult!
  - C.f. IPv6…

IP (the waist) is connectionless

- But most services above it are connection-oriented!
  - TCP, HTTP, SMTP, Skype, BitTorrent, etc. etc.
- So how to set up a connection?
  I.e. how to do signaling?

- TCP uses in-band signaling
  - Sends SYN/ACK/SYNACK/RST/FIN on same channel

In-band vs. out of band signalling

Out-of-band signalling

- Alternative approach
- Use separate signaling channel
  - Bootstrapped at start-of-day
  - Analogous step required in IP: DHCP (or ARP)
- Common in connection-oriented networks
  - E.g. ATM, GSM, …
- Also, mostly used with hard-state protocols

Soft state and hard state
Soft state vs. Hard state

- **Hard state:**
  - Explicit creation and deletion of state
  - No periodic updates necessary
  - Examples:
    - TCP connection setup (sort of)
    - NFS file handles
- **Soft state:**
  - Communicate state updates periodically
  - All information times out

**Hard state**

- “create”
- “update”
- “delete”

**Soft state**

- “update”
- “update”
- “update”

- Failure handled with no new functionality (or traffic)
- Failure mode is to remove unwanted state (eventually)
- Generally requires more bandwidth

**Examples:**
- Leases (DNS, DHCP, etc.)
- QoS Signaling (RSVP)
- Recovery and maintenance in robust DHTs (e.g. Bamboo)

**The End-to-End Argument**

- Simple, intuitive in non-failure case
- May have to do a lot of work in response to failure
- Can lead to instability
  - E.g. update storms when “failure” detected

**Examples:**
- TCP connection setup
- NFS file handles
- Voice circuit setup in GSM, etc.
The End-to-End argument: where to put network functionality

If a function can only be correctly implemented end-to-end, it must be implemented in the end systems. Implementing it in the network can, at best, only be an optimization.

• In these cases, end hosts:
  – Can provide the function without any network help
  – Must provide it without assuming any network help

Example: reliable file transfer

• This solution is not reliable!
  – One component fails (e.g. network corruption)
  – Others will not detect the failure
• Receiver still has to perform end-to-end check

Example: reliable file transfer

• Naïve solution:
  – Make every step reliable
  – Concatenate the steps into a single reliable transfer

Example: reliable file transfer

• Correct solution:
  – Both ends checksum the file on disk, and compare (or use secure hash, etc.)
  – If check fails, try again
  – Really matters even today (real disks, real Oses…)

End-to-End arguments

• Note that reliable transfer requires no network support at all!
• The Internet implements reliability in end hosts
  – TCP, etc.
• 802.11abgn also implements reliability at link layer
  – Q. Is this a good idea?
  – A. It might be for performance...
• The Internet implements routing in the network
  – But RON shows this is often slower

Other angles to E2E

• Sometimes, functions implemented in the network are bad for the application
  – E.g. reliable transfer for real-time traffic
• If function has to be end-to-end, implement it in host systems.
• Don’t put it in a lower layer except:
  – If it unambiguously improves performance
  – Does not burden applications that don’t need it
Robustness Principle

(Postel’s Law)

Be conservative in what you do,

be liberal in what you accept from others

Or

Be conservative in what you send,

liberal in what you accept

• Appeared in RFC 793, Transmission Control Protocol specification, September 1981

Example: Email

• The MIME standard RFCs define the correct format of a (multimedia) email message
• Questions:
  – Should you correctly format all email messages you send?
    • Yes!
  – Should you reject incorrectly formatted messages you receive?
    • No!
• Spam makes a great test for this

Example: HTTP

• Most HTTP/1.1 servers tolerate missing header files
  – Even when the standard requires them!

Robustness

• Much less likely that bugs in implementations prevent interoperability
  – Otherwise: minor bug $\Rightarrow$ total communication failure
• Easier to evolve protocols
  – Major version numbers $\Rightarrow$ incompatible, probably
  – Minor version numbers $\Rightarrow$ should be compatible
• Harder to program
  – Of course 😊

Fate sharing
Fate sharing

• To deal with potential failure:

  Store critical system state at the nodes which rely on that state

• Only way to lose that state is if the node that relies on it fails...
  ... in which case it doesn’t matter

Fate sharing example

• Goal: the only failure in the Internet which prevents communication should be total partition
  – If there exists a possible route, the Internet should find it

• Decision: Where to store connection state? I.e.
  – Flow control
  – Retransmission information
  – Etc.

Fate sharing example

• Flow-control state in end-systems:
  – if end-systems fail, connection state is irrelevant anyway

Fate sharing example

• Flow-control state in end-systems:
  – if end-systems fail, connection state is irrelevant anyway

Alternative: replication

• Maintain flow control state in routers
  – Requires replication and consistency
  – Failure of router requires state to be recreated
Fate sharing vs. replication

• Two key advantages:
  1. Easier to implement (no complex replication protocols)
  2. Resilient to any number of failures

• Other examples:
  – HTTP cookies
  – Packet-switching vs. circuit-switching

• Compare with End-to-End arguments

Alternative: replication

• Maintain flow control state in routers
  – Requires replication and consistency
  – Failure of router requires state to be recreated

References