Last week on
Communication Networks
Communication Networks
Part 1: General overview

#1 What is a network made of?

How is it shared?

How is it organized?

How does communication happen?

How do we characterize it?
Networks are composed of three basic components:

- **end-systems**
- **switch/routers**
- **links**
Communication Networks

Part 1: General overview

What is a network made of?

#2 How is it shared?

How is it organized?

How does communication happen?

How do we characterize it?
There exist two approaches to sharing: reservation and on-demand.

- **Reservation**: reserve the bandwidth you need in advance.
- **On-demand**: send data when you need it.
In practice, the approaches are implemented using circuit-switching or packet-switching
Pros and cons of circuit switching

<table>
<thead>
<tr>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Predictable performance</td>
<td>Inefficient if traffic is bursty or short</td>
</tr>
<tr>
<td>Simple &amp; fast switching</td>
<td>Complex circuit setup/teardown</td>
</tr>
<tr>
<td>Once circuit established</td>
<td>Which adds delays to transfer</td>
</tr>
<tr>
<td></td>
<td>Requires new circuit upon failure</td>
</tr>
</tbody>
</table>
Pros and cons of **packet switching**

<table>
<thead>
<tr>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>efficient use of resources</td>
<td>unpredictable performance</td>
</tr>
<tr>
<td>simpler to implement than circuit switching</td>
<td>requires buffer management and congestion control</td>
</tr>
<tr>
<td>route around trouble</td>
<td></td>
</tr>
</tbody>
</table>
Communication Networks

Part 1: General overview

What is a network made of?

How is it shared?

#3 How is it organized?

How does communication happen?

How do we characterize it?
This week on

Communication Networks
Communication Networks

Part 1: General overview

What is a network made of?

How is it shared?

How is it organized?

#4 How does communication happen?

#5 How do we characterize it?
Communication Networks

Part 1: General overview

What is a network made of?

How is it shared?

How is it organized?

How does communication happen?

How do we characterize it?
The Internet should allow

processes on different hosts
to exchange data

everything else is just commentary…
How do you exchange data in a network as complex as this?

http://www.opte.org
To exchange data, Alice and Bob use a set of network protocols
A protocol is like a conversational convention: who should talk next and how they should respond

give me http://comm-net.ethz.ch/

here it is
Sometimes implementations are not compliant…
Each protocol is governed by a specific interface

```
while (...) {
    message = ...;
    send(message, ...);
}
```

```
while (...) {
    message = receive(...);
}
```

Alice

Bob

**WoW server**

**WoW client**

**Application Programming Interface**
In practice, there exists a lot of network protocols. How does the Internet organize this?
How standards proliferate:

(See: A/C chargers, character encodings, instant messaging, etc.)

**Situation:**
There are 14 competing standards.

14?! Ridiculous!
We need to develop
one universal standard
that covers everyone's
use cases.

Yeah!

**Soon:**

**Situation:**
There are 15 competing standards.

https://xkcd.com/927/
Modularity is a key component of any good system

Problem

- can’t build large systems out of spaghetti code
- hard (if not, impossible) to understand, debug, update
- need to bound the scope of changes
- evolve the system without rewriting it from scratch

Solution

- Modularity is how we do it
- …and understand the system at a higher-level
Modularity, based on abstraction, is *the* way things get done

— Barbara Liskov, MIT
To provide structure to the design of network protocols, network designers organize **protocols** in layers and the network hardware/software that implement them.
Internet communication can be decomposed in **5 independent layers** (or 7 layers for the OSI model)

<table>
<thead>
<tr>
<th>Layer</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>L5</td>
<td>Application</td>
</tr>
<tr>
<td>L4</td>
<td>Transport</td>
</tr>
<tr>
<td>L3</td>
<td>Network</td>
</tr>
<tr>
<td>L2</td>
<td>Link</td>
</tr>
<tr>
<td>L1</td>
<td>Physical</td>
</tr>
</tbody>
</table>
Each layer provides a service to the layer above

<table>
<thead>
<tr>
<th>Layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>L5</td>
</tr>
<tr>
<td>L4</td>
</tr>
<tr>
<td>L3</td>
</tr>
<tr>
<td>L2</td>
</tr>
<tr>
<td>L1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Service Provided</th>
</tr>
</thead>
<tbody>
<tr>
<td>network access</td>
</tr>
<tr>
<td>end-to-end delivery (reliable or not)</td>
</tr>
<tr>
<td>global best-effort delivery</td>
</tr>
<tr>
<td>local best-effort delivery</td>
</tr>
<tr>
<td>physical transfer of bits</td>
</tr>
</tbody>
</table>
Each layer provides a service to the layer above by using the services of the layer directly below it.

- **Applications**
  - ...built on...

- **Reliable (or unreliable) transport**
  - ...built on...

- **Best-effort global packet delivery**
  - ...built on...

- **Best-effort local packet delivery**
  - ...built on...

- **Physical transfer of bits**
Each layer has a unit of **data**

<table>
<thead>
<tr>
<th>layer</th>
<th>role</th>
</tr>
</thead>
<tbody>
<tr>
<td>L5</td>
<td>Application exchanges <strong>messages</strong> between processes</td>
</tr>
<tr>
<td>L4</td>
<td>Transport transports <strong>segments</strong> between end-systems</td>
</tr>
<tr>
<td>L3</td>
<td>Network moves <strong>packets</strong> around the network</td>
</tr>
<tr>
<td>L2</td>
<td>Link moves <strong>frames</strong> across a link</td>
</tr>
<tr>
<td>L1</td>
<td>Physical moves <strong>bits</strong> across a physical medium</td>
</tr>
</tbody>
</table>
Each layer (except for L3) is implemented with different protocols

<table>
<thead>
<tr>
<th>Layer</th>
<th>Protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>L5</td>
<td>Application, HTTP, SMTP, FTP, SIP, ...</td>
</tr>
<tr>
<td>L4</td>
<td>Transport, TCP, UDP, SCTP</td>
</tr>
<tr>
<td>L3</td>
<td>Network, IP</td>
</tr>
<tr>
<td>L2</td>
<td>Link, Ethernet, Wifi, (A/V)DSL, WiMAX, LTE, ...</td>
</tr>
<tr>
<td>L1</td>
<td>Physical, Twisted pair, fiber, coaxial cable, ...</td>
</tr>
</tbody>
</table>
The Internet Protocol (IP) acts as an unifying, network, layer

<table>
<thead>
<tr>
<th>Layer</th>
<th>Protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>L5</td>
<td>Application: HTTP, SMTP, FTP, SIP, ...</td>
</tr>
<tr>
<td>L4</td>
<td>Transport: TCP, UDP, SCTP</td>
</tr>
<tr>
<td>L3</td>
<td>Network: IP</td>
</tr>
<tr>
<td>L2</td>
<td>Link: Ethernet, Wifi, (A/V)DSL, Cable, LTE, ...</td>
</tr>
<tr>
<td>L1</td>
<td>Physical: Twisted pair, fiber, coaxial cable, ...</td>
</tr>
</tbody>
</table>
Each layer is implemented with different protocols and technologies
hardware

L1  L2  L3  L4  L5

speed

flexibility

software
Software and hardware advancements

- Hardware
  - Speed
  - Flexibility

- Software

Programmable network devices

Highly optimized libraries, drivers

SDN, P4

DPDK, FD.io
Microsoft Supercharges Bing Search With Programmable Chips

DOUG BURGER CALLED it Project Catapult.

Burger works inside Microsoft Research—the group where the tech giant explores blue-sky ideas—and in November 2012, he pitched a radical new concept to Qi Lu, the man who

https://www.wired.com/2014/06/microsoft-fpga/
Each layer takes messages from the layer above, and *encapsulates* with its own header and/or trailer.
Your laptop sends a request to Google using HTTP(S) protocol. The request includes a header and a message. The header contains a Host (HA) field with the value `google.ch`. The process involves the following layers:

1. **Application**
   - HTTP(S)
2. **Transport**
   - TCP/UDP
3. **Network**
   - IP
4. **Link**
   - Ethernet
In practice, layers are distributed on every network device.
Since when bits arrive they must make it to the application, all the layers exist on a host.
Routers act as **L3 gateway** as such they implement L2 and L3
Switches act as **L2 gateway** as such they only implement L2
Let’s see how it looks like in practice on a host, using Wireshark

https://www.wireshark.org
Communication Networks

Part 1: General overview

What is a network made of?

How is it shared?

How is it organized?

How does communication happen?

#5 How do we characterize it?
A network *connection* is characterized by its delay, loss rate and throughput.

How long does it take for a packet to reach the destination?

What fraction of packets sent to a destination are dropped?

At what rate is the destination receiving data from the source?
A network *connection* is characterized by its delay, loss rate and throughput.
Each packet suffers from several types of delays at each node along the path

\[
\text{transmission delay} \quad \oplus \quad \text{propagation delay} \quad \oplus \quad \text{processing delay} \quad \oplus \quad \text{queuing delay} = \text{total delay}
\]

\[\text{due to link properties}\]
\[\text{due to traffic mix & switch internals}\]
Overall, the main culprits for the overall delay are the transmission, propagation and queuing delays tend to be tiny.

\[
\text{transmission delay} + \text{propagation delay} + \text{processing delay} + \text{queuing delay} = \text{total delay}
\]
The transmission delay is the amount of time required to push all of the bits onto the link.

Transmission delay = \( \frac{\text{packet size}}{\text{link bandwidth}} \) [sec]

Example

\[
\frac{1000 \text{ bits}}{100 \text{ Gbps}} = 10 \text{ ns}
\]
The propagation delay is the amount of time required for a bit to travel to the end of the link.

\[
\text{Propagation delay} = \frac{\text{link length}}{\text{propagation speed}} \quad [\text{sec}]
\]

**Example**

\[
\begin{align*}
\text{link length} &= 30\,000\,\text{m} \\
\text{propagation speed} &= 2\times10^8\,\text{m/sec} \\
\text{Propagation delay} &= \frac{30\,000\,\text{m}}{2\times10^8\,\text{m/sec}} = 150\,\mu\text{sec}
\end{align*}
\]
How long does it take for a packet to travel from A to B?
(not considering queuing for now)
How long does it take to exchange 100 Bytes packet?

Time to transmit one bit = $10^{-6}$s

Time to transmit 800 bits = $800 \times 10^{-6}$s

Time when that bit reaches B: $10^{-6} + 10^{-3}$s

The last bit reaches B at $(800 \times 10^{-6}) + 10^{-3}$s = 1.8ms
If we have a 1 Gbps link, the total time decreases to 1.0008ms.
If we now exchange a 1GB file split in 100B packets

\[ 10^7 \times 100B \text{ packets} \]

The last bit reaches B at
\[ (10^7 \times 800 \times 10^{-9}) + 10^{-3}s \]
\[ = 8001ms \]
Different transmission characteristics imply different tradeoffs in terms of which delay dominates.

- $10^7 \times 100B$ pkt 1Gbps link: transmission delay dominates.
- $1 \times 100B$ pkt 1Gbps link: propagation delay dominates.
- $1 \times 100B$ pkt 1Mbps link: both matter.

In the Internet, we can't know in advance which one matters!
The queuing delay is the amount of time a packet waits (in a buffer) to be transmitted on a link.

Queuing delay is the hardest to evaluate as it varies from packet to packet.

It is characterized with statistical measures e.g., average delay & variance, probability of exceeding $x$. 
Queuing delay depends on the traffic pattern
Queuing delay depends on the traffic pattern

Queue

Transient overload!
Transient overload!
Queues absorb transient bursts, but introduce queueing delays
The time a packet has to sit in a buffer before being processed depends on the traffic pattern.

Queueing delay depends on:

- arrival rate at the queue
- transmission rate of the outgoing link
- traffic burstiness
average packet arrival rate $\lambda$ [packet/sec]

transmission rate of outgoing link $R$ [bit/sec]

fixed packets length $L$ [bit]

average bits arrival rate $La$ [bit/sec]

traffic intensity $La/R$
When the traffic intensity is $>1$, the queue will increase without bound, and so does the queuing delay

Golden rule

Design your queuing system, so that it operates far from that point.
When the traffic intensity is <=1, queueing delay depends on the burst size.
A network *connection* is characterized by its delay, loss rate and throughput.
In practice, queues are not infinite. There is an upper bound on queuing delay.

\[ \text{queuing delay upper bound: } \frac{N \times L}{R} \]
If the queue is persistently overloaded, it will eventually drop packets (loss)
A network *connection* is characterized by its delay, loss rate and throughput.
The throughput is the instantaneous rate at which a host receives data

\[
\text{Average throughput} = \frac{\text{data size}}{\text{transfer time}} \quad [\#\text{bits/sec}]
\]
To compute throughput, one has to consider the bottleneck link

Average throughput

\[ \text{min}(R_S, R_L) \]

= transmission rate of the bottleneck link

\[ F \quad \text{file size} \]
To compute throughput, one has to consider the bottleneck link... and the intervening traffic

If $4 \times \min(R_S, R_L) > R$, the bottleneck is now in the core, providing each download $R/4$ of throughput.
A network *connection* is characterized by its delay, loss rate and throughput.
As technology improves, throughput increase & delays are getting lower except for propagation (speed of light)

source: ciena.com
Because of propagation delays, Content Delivery Networks move content closer to you

http://wwwnui.akamai.com/gnet/globe/index.html
Communication Networks

Part 1: General overview

What is a network made of?

How is it shared?

How is it organized?

How does communication happen?

How do we characterize it?
Communication Networks
Part 2: Concepts

routing

reliable delivery
How do you guide IP packets from a source to destination?

How do you ensure reliable transport on top of best-effort delivery?
How do you guide IP packets from a source to destination?
Think of IP packets as envelopes

Packet
Like an envelope, packets have a header.
Like an envelope, packets have a payload
The header contains the metadata needed to forward the packet.

Identify the source and destination of the communication.

- src address
- dst address

- source
- destination
The payload contains the data to be delivered.
Routers forward IP packets *hop-by-hop* towards their destination.
Let’s zoom in on what is going on between two adjacent routers
Upon packet reception, routers **locally** look up their forwarding table to know where to send it next.
Here, the packet should be directed to **IF#4**
Forwarding is repeated at each router, until the destination is reached.
Forwarding table

<table>
<thead>
<tr>
<th>src</th>
<th>destination</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Laurent</td>
<td>Google</td>
<td>IF#1</td>
</tr>
<tr>
<td>Google</td>
<td></td>
<td>IF#3</td>
</tr>
</tbody>
</table>
Forwarding decisions necessarily depend on the destination, but can also depend on other criteria.

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Destination</th>
<th>Source</th>
<th>Input Port</th>
<th>Any Other Header</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>mandatory</td>
<td>requires $n^2$ state</td>
<td>traffic engineering</td>
<td></td>
</tr>
</tbody>
</table>
In the rest of the lecture, we’ll consider destination-based routing—the default in the Internet.
Where are these forwarding tables coming from?

<table>
<thead>
<tr>
<th>Destination</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Laurent</td>
<td>IF#1</td>
</tr>
<tr>
<td>Google</td>
<td>IF#4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Destination</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Laurent</td>
<td>IF#1</td>
</tr>
<tr>
<td>Google</td>
<td>IF#3</td>
</tr>
</tbody>
</table>
In addition to a data-plane, routers are also equipped with a control-plane.
Think of the control-plane as the router’s brain

Roles

Routing

Configuration

Statistics

...
Routing is the control-plane process that computes and populates the forwarding tables.

Control-Plane

- Destination: Laurent
  - Output: IF#1
- Destination: Google
  - Output: IF#4

Control-Plane

- Destination: Laurent
  - Output: IF#1
- Destination: Google
  - Output: IF#3
How can a router know where to direct packets if it does not know what the network looks like?

While forwarding is a local process, routing is inherently a global process.
## Forwarding vs Routing

### Summary

<table>
<thead>
<tr>
<th>forwarding</th>
<th>routing</th>
</tr>
</thead>
<tbody>
<tr>
<td>goal</td>
<td>directing packet to an outgoing link</td>
</tr>
<tr>
<td>scope</td>
<td>local</td>
</tr>
<tr>
<td>implem.</td>
<td>hardware usually</td>
</tr>
<tr>
<td>timescale</td>
<td>nanoseconds</td>
</tr>
</tbody>
</table>
The goal of routing is to compute valid global forwarding state.

Definition: a global forwarding state is valid if it always delivers packets to the correct destination.
Theorem

A global forwarding state is valid if and only if

- there are no dead ends
- no outgoing port defined in the table
- there are no loops
- packets going around the same set of nodes
A global forwarding state is valid if and only if there are no dead ends.
A global forwarding state is valid if and only if there are no forwarding loops.
Proving the necessary condition is easy

Theorem
If a routing state is valid
then there are no loops or dead-end

Proof
If you run into a dead-end or a loop
you’ll never reach the destination
so the state cannot be correct (contradiction)
Proving the **sufficient** condition is more subtle

**Theorem**  
If a routing state has no dead end and no loop then it is valid

**Proof**  
There is only a finite number of ports to visit

A packet can never enter a switch via the same port, otherwise it is a loop *(which does not exist by assumption)*

As such, the packet must **eventually** reach the destination
question 1  How do we verify that a forwarding state is valid?

question 2  How do we compute valid forwarding state?
question 1

How do we verify that a forwarding state is valid?

How do we compute valid forwarding state?
Verifying that a routing state is valid is easy

- simple algorithm
  - for one destination
- Mark all outgoing ports with an arrow
- Eliminate all links with no arrow
- State is valid iff the remaining graph is a spanning-tree
Given a graph with the corresponding forwarding state
Mark all outgoing ports with an arrow
Eliminate all links with no arrow
The result is a spanning tree.
This is a valid routing state
Mark all outgoing ports with an arrow
Eliminate all links with no arrow
The result is **not a spanning-tree.**
The routing state is **not valid**
question 2

How do we verify that a forwarding state is valid?

How do we compute valid forwarding state?
Producing valid routing state is harder

- Prevent dead ends: Easy
- Prevent loops: Hard
Producing valid routing state is harder but doable.

- Prevent dead ends: easy
- Prevent loops: hard

This is the question you should focus on.
Existing routing protocols differ in how they avoid loops
Essentially, there are three ways to compute valid routing state:

<table>
<thead>
<tr>
<th>Intuition</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1 Use tree-like topologies</td>
<td>Spanning-tree</td>
</tr>
<tr>
<td>#2 Rely on a global network view</td>
<td>Link-State SDN</td>
</tr>
<tr>
<td>#3 Rely on distributed computation</td>
<td>Distance-Vector BGP</td>
</tr>
</tbody>
</table>