Communication Networks
Spring 2020

Laurent Vanbever
nsg.ee.ethz.ch

ETH Zürich (D-ITET)
March 9 2020

Materials inspired from Scott Shenker & Jennifer Rexford
Please register your group for the projects
https://comm-net.ethz.ch by Friday
Last week on Communication Networks
We started looking at the two fundamental challenges underlying networking:

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?
Routing is the control-plane process that computes and populates the forwarding tables.
# Forwarding vs Routing

## Summary

<table>
<thead>
<tr>
<th></th>
<th>Forwarding</th>
<th>Routing</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>goal</strong></td>
<td>Directing packet to an outgoing link</td>
<td>Computing the paths packets will follow</td>
</tr>
<tr>
<td><strong>scope</strong></td>
<td>Local</td>
<td>Network-wide</td>
</tr>
<tr>
<td><strong>implem.</strong></td>
<td>Hardware</td>
<td>Software</td>
</tr>
<tr>
<td></td>
<td>Usually</td>
<td>Always</td>
</tr>
<tr>
<td><strong>timescale</strong></td>
<td>Nanoseconds</td>
<td>10s of ms</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Hopefully</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
How do we verify that a forwarding state is valid?

How do we compute valid forwarding state?
Producing valid routing state is harder, but doable.

- Prevent dead ends: easy
- Prevent loops: hard

This is the question you should focus on.
You acted as IP routers and run a flavor of distributed Breadth-First-Search (BFS)
We then looked at the **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</td>
</tr>
<tr>
<td>#3 Rely on distributed computation</td>
<td>SDN</td>
</tr>
<tr>
<td></td>
<td>Distance-Vector</td>
</tr>
<tr>
<td></td>
<td>BGP</td>
</tr>
<tr>
<td>#1</td>
<td>Use tree-like topologies</td>
</tr>
<tr>
<td>----</td>
<td>-------------------------</td>
</tr>
<tr>
<td></td>
<td>Rely on a global network view</td>
</tr>
<tr>
<td></td>
<td>Rely on distributed computation</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The easiest way to avoid loops is to route traffic on a loop-free topology

simple algorithm

Take an arbitrary topology
Build a spanning tree and ignore all other links

Done!

Why does it work?
Spanning-trees have only one path between any two nodes
Use tree-like topologies

Rely on a global network view

Rely on distributed computation
If each router knows the entire graph, it can locally compute paths to all other nodes.
Once a node $u$ knows the entire topology, it can compute shortest-paths using Dijkstra’s algorithm.

**Initialization**

$S = \{u\}$

*for all nodes $v$:*

  *if ($v$ is adjacent to $u$):*

    $D(v) = c(u, v)$

  *else:*

    $D(v) = \infty$

**Loop**

*while not all nodes in $S$:*

  *add $w$ with the smallest $D(w)$ to $S*

  *update $D(v)$ for all adjacent $v$ not in $S:*

    $D(v) = \min\{D(v), D(w) + c(w, v)\}$
Essentially, there are three ways to compute valid routing state:

- Use tree-like topologies: Spanning-tree
- Rely on a global network view: Link-State, SDN
- #3 Rely on distributed computation: Distance-Vector, BGP
Let $d(x(y))$ be the cost of the least-cost path known by $x$ to reach $y$.

Each node bundles these distances into one message (called a vector) that it repeatedly sends to all its neighbors until convergence.

Each node updates its distances based on neighbors’ vectors:

$$d(x(y)) = \min \{ c(x,v) + d(v(y)) \} \text{ over all neighbors } v$$
This week on Communication Networks
This week we'll start speaking about

How the Internet actually works
We’ll do that layer-by-layer, bottom-up, starting with the Link layer.
How do local computers communicate?
Communication Networks
Part 2: The Link Layer

#1 What is a link?
#2 How do we identify link adapters?
#3 How do we share a network medium?
#4 What is Ethernet?
#5 How do we interconnect segments at the link layer?
Communication Networks
Part 2: The Link Layer

#1 What is a link?

How do we identify link adapters?

How do we share a network medium?

What is Ethernet?

How do we interconnect segments at the link layer?
Link  Communication medium  and  Network adapter

- Wifi
- Ethernet
- Fiber
Network adapters communicate together through the medium
Network adapters communicate together through the medium. 

Sender:
- Encapsulate packets in a frame
- Add error checking bits, flow control, ...

Receiver:
- Look for errors, flow control, ...
- Extract packet and passes it to the network layer
The Link Layer provides a best-effort delivery service to the Network layer.

<table>
<thead>
<tr>
<th>L3</th>
<th>Network</th>
<th>global best-effort delivery</th>
</tr>
</thead>
<tbody>
<tr>
<td>L2</td>
<td>Link</td>
<td>local best-effort delivery</td>
</tr>
<tr>
<td>L1</td>
<td>Physical</td>
<td>physical transfer of bits</td>
</tr>
</tbody>
</table>
The Link Layer provides a best-effort delivery service to the Network layer, composed of 5 sub-services:

- **encoding**: represents the 0s and the 1s
- **framing**: encapsulate packet into a frame, adding header and trailer
- **error detection**: detects errors with checksum
- **error correction**: optionally correct errors
- **flow control**: pace sending and receiving node
As of March 2020, State-of-the-art Ethernet adapters clock at 200 Gbps.

- 215 million pkt/sec
- sub 0.8 usec latency
- PCIe Gen 4.0

source: [Mellanox ConnectX-6]
What is a link?

#2 How do we identify link adapters?

How do we share a network medium?

What is Ethernet?

How do we interconnect segments at the link layer?
Medium Access Control addresses
MAC addresses...
MAC addresses…

identify the sender & receiver adapters
used within a link

are uniquely assigned
hard-coded into the adapter when built

use a flat space of 48 bits
allocated hierarchically
MAC addresses are hierarchically allocated

34:36:3b:d2:8a:86
The **first** 24 bits blocks are assigned to network adapter vendor by the IEEE

Apple, Inc.
1 Infinite Loop
Cupertino CA 95014
US

see http://standards-oui.ieee.org/oui/oui.txt
The **second** 24 bits block is assigned by the vendor to each network adapter.

34:36:3b:d2:8a:86

assigned by Apple to my adapter
The address with all bits set to 1 identifies the broadcast address

\texttt{ff:ff:ff:ff:ff:ff}

enables to send a frame to all adapters on the link
By default, adapters only decapsulates frames addressed to the local MAC or the broadcast address.
The promiscuous mode enables to decapsulate *everything*, independently of the destination MAC.
Why don’t we simply use IP addresses?

Links can support any protocol (not just IP) different addresses on different kind of links

Adapters may move to different locations cannot assign static IP address, it has to change

Adapters must be identified during bootstrap need to talk to an adapter to give it an IP address
Adapters must be identified during bootstrap

need to talk to an adapter to give it an IP address
You need to solve two problems when you bootstrap an adapter.

**Who am I?**
- MAC-to-IP binding

**How do I acquire an IP address?**

**Who are you?**
- IP-to-MAC binding

**Given an IP address reachable on a link,**
- How do I find out what MAC to use?
Who am I?
MAC-to-IP binding

Who are you?
IP-to-MAC binding

How do I acquire an IP address?
Dynamic Host Configuration Protocol

Given an IP address reachable on a link,
How do I find out what MAC to use?
Address Resolution Protocol
Network adapters traditionally acquire an IP address using the Dynamic Host Configuration Protocol (DHCP)
Every connected device needs an IP address...
no ip :(
Host sends an “IP request” to everyone on the link using the broadcast address.

34:36:3b:d2:8a:10
no ip :(

34:36:3b:d2:8a:89
192.168.1.1

34:36:3b:d2:8a:86
192.168.1.10
DHCP discovery

- **dstmac**: ff:ff:ff:ff:ff:ff
- **payload**: I want an IP

---

34:36:3b:d2:8a:10
no ip :(

---

34:36:3b:d2:8a:89
192.168.1.1

---

34:36:3b:d2:8a:86
192.168.1.10
DHCP server (if any) answers with an IP address

34:36:3b:d2:8a:10
no ip :(  

34:36:3b:d2:8a:89
192.168.1.1

34:36:3b:d2:8a:86
192.168.1.10
DHCP offer

dstmac 34:36:3b:d2:8a:10
payload use 192.168.1.9

34:36:3b:d2:8a:10
no ip :(

34:36:3b:d2:8a:86
192.168.1.10

34:36:3b:d2:8a:89
192.168.1.1
The Address Resolution Protocol (ARP) enables a host to discover the MAC associated to an IP.
I want to send an IP packet to 192.168.1.10? What destination MAC do I use?!
Who has 192.168.1.10?
Tell 192.168.1.9

ARP request

dstmac  ff:ff:ff:ff:ff:ff
payload  Who has 192.168.1.10?
          Tell 192.168.1.9

34:36:3b:d2:8a:10  192.168.1.9
34:36:3b:d2:8a:89  192.168.1.1
34:36:3b:d2:8a:86  192.168.1.10
ARP reply

**dstmac** 34:36:3b:d2:8a:10

**payload** 192.168.1.10 is at

34:36:3b:d2:8a:86

34:36:3b:d2:8a:10
192.168.1.9

34:36:3b:d2:8a:86
192.168.1.10
ARP table

- 192.168.1.10: 34:36:3b:d2:8a:86
- 192.168.1.9: 34:36:3b:d2:8a:10
- 192.168.1.1: 34:36:3b:d2:8a:89

Diagram:

- Mac mini: 34:36:3b:d2:8a:10
  - 192.168.1.9
- Apple Thunderbolt Display: 34:36:3b:d2:8a:86
  - 192.168.1.10
- Mac Pro: 34:36:3b:d2:8a:89
  - 192.168.1.1
What is a link?

How do we identify link adapters?

#3 How do we share a network medium?

What is Ethernet?

How do we interconnect segments at the link layer?
Some medium are **multi-access**: 

>1 host can communicate at the same time
Some medium are **multi-access**: >1 host can communicate at the same time

- Wireless networks
- Satellite networks
- Ethernet networks
- Cellular networks
Some medium are **multi-access**: >1 host can communicate at the same time

<table>
<thead>
<tr>
<th>Problem</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>collisions lead to garbled data</td>
<td>distributed algorithm for sharing the channel</td>
</tr>
</tbody>
</table>

When can each node transmit?
Essentially, there are three techniques to deal with Multiple Access Control (MAC):

- **Divide the channel into pieces**
  either in time or in frequency

- **Take turns**
  pass a token for the right to transmit

- **Random access**
  allow collisions, detect them and then recover
What is a link?

How do we identify link adapters?

How do we share a network medium?

What is Ethernet?

How do we interconnect segments at the link layer?
Ethernet...

was invented as a broadcast technology
each packet was received by all attached hosts

is now the dominant wired LAN technology
by far the most widely used

has managed to keep up with the speed race
from 10 Mbps to 400 Gbps (next goal: 1 Tbps!)
Ethernet offers an unreliable, connectionless service

unreliable

Receiving adapter does not acknowledge anything

Packets passed to the network layer can have gaps which can be filled by the transport protocol (TCP)

connectionless

No handshaking between the send and receive adapter
“Traditional” Ethernet relies on CSMA/CD
CSMA/CD imposes limits on the network length

Suppose A sends a packet at time \( t \)

B sees an idle line just before \( t+d \) and sends a packet

Effect

B would detect a collision and sends a jamming signal

A can detect the collision only after \( t+2d \)
For this reason, Ethernet imposes a minimum packet size (512 bits)

This imposes restriction on the length of the network

\[
\text{Network length} = \frac{\text{min\_frame\_size} \times \text{speed of light}}{2 \times \text{bandwidth}}
\]

\[
= 768 \text{ meters} \quad \text{for 100 Mbps}
\]

What about for 1 Gbps, 10 Gbps, 100 Gbps?
Modern Ethernet links interconnects *exactly* two hosts, in full-duplex, rendering collisions impossible!

CSMA/CD is only needed for half-duplex communications. 10 Gbps Ethernet does not even allow half-duplex anymore.

This means the 64 bytes restriction is not strictly needed, but IEEE chose to keep it.

Multiple Access Protocols are still important for Wireless, important concepts to know in practice.
The Ethernet header is simple, composed of 6 fields only:

- **preamble**: used for synchronization
- **dest address**: usually, IPv4 (0x0800)
- **src address**:
- **type**
- **data**
- **CRC**: Cyclic Redundant Check
Ethernet efficiency (payload/tot. frame size): \(~97.5\%\)

Maximum throughput for 100 Mbps: \(~97.50\) Mbps
What is a link?

How do we identify link adapters?

How do we share a network medium?

What is Ethernet?

How do we interconnect segments at the link layer?
Historically, people connected Ethernet segments together at the physical level using **Ethernet hubs**.
Hubs work by repeating bits from one port to all the other ones.
Hubs are now obsolete

advantages
simple, cheap

disadvantages
inefficient, each bit is sent everywhere
limits the aggregates throughput
limited to one LAN technology
can’t interconnect different rates/formats
limited number of nodes and distances
cannot go beyond 2500m on Ethernet
Local Area Networks are now almost exclusively composed of Ethernet switches
Switches connect two or more LANs together at the Link layer, acting as L2 gateways.

Switches are “store-and-forward” devices, they:

- extract the destination MAC from the frame
- look up the MAC in a table (using exact match)
- forward the frame on the appropriate interface

Switches are similar to IP routers, except that they operate one layer below.
Unlike with hubs, switches enable each LAN segment to carry its own traffic.
Unlike with hubs, switches support concurrent communication.

B and F can talk to each other, while A and C are talking.
The advantages of switches are numerous

advantages

only forward frames where needed
avoids unnecessary load on segments

join segment using different technologies

improved privacy
host can just snoop traffic traversing their segment

wider geographic span
separates segments allow longer distance
Switches are plug-and-play devices, they build their forwarding table on their own.
Switches are “store-and-forward” devices, they

- extract the destination MAC from the frame
- look up the MAC in a table (using exact match)
- forward the frame on the appropriate interface
Switches are plug-and-play devices, they build their forwarding table on their own.

When a frame arrives:
- inspect the source MAC address
- associate the address with the port
- store the mapping in the switch table
- launch a timer to eventually forget the mapping

switch learns how to reach A
In cases of misses, switches simply floods the frames

When a frame arrives with **an unknown destination**

- forward the frame **out of all interfaces** except for the one where the frame arrived

Hopefully, this is an unlikely event

when in doubt, **shout!**
While flooding enables automatic discovery of hosts, it also creates problems when the network has loops.

Each frame leads to the creation of at least two new frames! An exponential increase, with no TTL to remove looping frames...
While loops create major problems, networks need redundancy for tolerating failures!

**Solution**

Reduce the network to one logical spanning tree.

Upon failure, automatically rebuild a spanning tree.
In practice, switches run a *distributed* Spanning-Tree Protocol (STP)
I think that I shall never see
A graph more lovely than a tree.
A tree whose crucial property
Is loop-free connectivity.

A tree that must be sure to span
So packets can reach every LAN.
First, the root must be selected.
By ID, it is elected.

Least-cost paths from root are traced.
In the tree, these paths are placed.
A mesh is made by folks like me,
Then bridges find a spanning tree.

— Radia Perlman
A tree that must be sure to span
So packets can reach every LAN.
First, the root must be selected.
By ID, it is elected.

Least-cost paths from root are traced.
In the tree, these paths are placed.
A mesh is made by folks like me,
Then bridges find a spanning tree.
Constructing a Spanning Tree in a nutshell

Switches...

elect a root switch
the one with the smallest identifier

determine if each interface is
on the shortest-path from the root
and disable it if not
For this switches exchange
Bridge Protocol Data Unit (BDPU) messages

Each switch $X$ iteratively sends

$$\text{BPDU} \ (Y, \ d, \ X)$$

to each neighboring switch

the switch ID it considers as root

the # hops to reach it
initially

Each switch proposes itself as root
sends $(X,0,X)$ on all its interfaces

Upon receiving $(Y, d, X)$, checks if $Y$ is a better root
if so, considers $Y$ as the new root, flood updated message

Switches compute their distance to the root, for each port
simply add 1 to the distance received, if shorter, flood

Switches disable interfaces not on shortest-path
Upon receiving ≠ BPDUs from ≠ switches with ≠ cost
Pick the BPDU with the lower switch sender ID

Upon receiving ≠ BPDUs from a neighboring switch
Pick the BPDU with the lowest port ID (e.g. port 2 < port 3)
Apply the algorithm starting with switch 4
Apply the algorithm starting with switch 4
To be robust, STP must react to failures

Any switch, link or port can fail
including the root switch

Root switch continuously sends messages
announcing itself as the root (1,0,1), others forward it

Failures is detected through timeout (soft state)
if no word from root in X, times out and claims to be the root