Communication Networks

Prof. Laurent Vanbever

Last week on 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?

#4 How does communication happen?

In practice, layers are distributed on every network device

HTTP(S)
TCP/UDP
IP
Ethernet
IP
HTTP(S)
TCP/UDP
IP
Ethernet

Material inspired from Scott Shenker & Jennifer Rexford
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?

Each packet suffers from several types of delays at each node along the path

transmission delay
processing delay
queueing delay
= total delay
due to link properties
due to traffic mix & switch internals

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

If the queue is persistently overloaded, it will eventually drop packets (loss)
To compute throughput, one has to consider the bottleneck link

\[ \text{Average throughput} = \min(R_s, R_l) \]

Where:
- \( R_s \): transmission rate of the server
- \( R_l \): transmission rate of the client

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?

We will dive into two fundamental networking challenges

- Routing
- Reliable delivery

This week on Communication Networks

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

Next week

Like an envelope, packets have a header

Think of IP packets as envelopes

Packet

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.

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 decisions necessarily depend on the destination, but can also depend on other criteria:

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Destination</td>
<td>mandatory (why?)</td>
</tr>
<tr>
<td>Source</td>
<td>requires $n^2$ state</td>
</tr>
<tr>
<td>Input Port</td>
<td>traffic engineering</td>
</tr>
<tr>
<td>Other Header</td>
<td></td>
</tr>
</tbody>
</table>

With source- & destination-based routing, paths from different sources can differ:

Let’s compare these two:

With destination-based routing, paths from different sources coincide once they overlap:

Once path to destination meet, they will never split:

Set of paths to the destination produce a spanning tree rooted at the destination:

- cover every router exactly once
- only one outgoing arrow at each router
Here is an example of a spanning tree for destination $X$.

In the rest of the lecture, we'll consider destination-based routing, the default in the Internet.

Where are these forwarding tables coming from?

In addition to the data-plane, routers are also equipped with a control-plane.

Think of the control-plane as the router's brain.

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

While forwarding is a local process, routing is inherently a global process.

How can a router know where to direct packets if it does not know what the network looks like?
Forwarding vs Routing

**Summary**

<table>
<thead>
<tr>
<th>Forwarding</th>
<th>Routing</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Goal</strong></td>
<td>directing packet to an outgoing link</td>
</tr>
<tr>
<td><strong>Scope</strong></td>
<td>local</td>
</tr>
<tr>
<td><strong>Implementation</strong></td>
<td>hardware</td>
</tr>
<tr>
<td><strong>Timescale</strong></td>
<td>nanoseconds</td>
</tr>
</tbody>
</table>

The goal of routing is to compute a valid global forwarding state.

A global forwarding state is valid if and only if:

- there are no dead ends
- there are no loops

No dead ends and no loops are a sufficient and necessary condition for forwarding validity.

To prove statement 3, we must prove statement 1 and statement 2.

Statement 1: A if B means B implies A
- if B is true, then A is true

Statement 2: A only if B means A implies B
- if A is true, then B is true

Statement 3: A if and only if B means both
- if A is true, then so is B and vice-versa

Theorem:

A global forwarding state is valid if and only if:

- there are no dead ends
- there are no loops

Sufficient and necessary condition:

- if A is true, then so is B and vice-versa
- i.e. packets going around the same set of nodes
- i.e. no outgoing port defined in the table

Statement 1

Statement 2

Statement 3
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?

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

Producing valid routing state is harder, but doable

How do we verify that a forwarding state is valid?

How do we compute valid forwarding state?

Produce valid routing state is harder

prevent dead ends

easy

prevent loops

hard

This is the question you should focus on
Most routing protocols out there 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>

The easiest way to avoid loops is to use a topology where loops are impossible.

**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.

Before I give you all the answers, it’s your turn...

...to figure out a way to route traffic in a network.

Instructions given in class.

In practice, there can be many spanning-trees for a given topology.

Spanning-Tree #1

Spanning-Tree #2
We’ll see how to compute a spanning-tree in two weeks. For now, assume it is possible.

Once we have a spanning tree, forwarding on it is easy. Literally just flood the packets everywhere.

When a packet arrives, simply send it on all ports.

While flooding works, it is quite wasteful.

Luckily, nodes can learn how to reach nodes by remembering where packets came from.

```
intuition
if
  flood packet from node A
  entered switch X on port 4
then
  switch X can use port 4
  to reach node A
```
All the green nodes learn how to reach A

All the nodes in the network know on which port A can be reached

Now B answers back to A enabling the green nodes to also learn where B is

There is no need for flooding here as the position of A is already known by everybody

Routing by flooding on a spanning-tree in a nutshell

Spanning-Tree in practice used in Ethernet

<table>
<thead>
<tr>
<th>advantages</th>
<th>disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>plug-and-play</td>
<td>mandate a spanning-tree</td>
</tr>
<tr>
<td>configuration-free</td>
<td>eliminate many links from the topology</td>
</tr>
<tr>
<td>automatically adapts to moving host</td>
<td>slow to react to failures</td>
</tr>
<tr>
<td></td>
<td>host movement</td>
</tr>
</tbody>
</table>

In practice, operators tend to dislike Spanning Tree…
Essentially, there are three ways to compute valid routing state:

1. Use tree-like topologies
2. Rely on a global network view
3. Rely on distributed computation

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 $S$:
    - $D(v) = \min\{D(v), D(w) + c(w,v)\}$

Let’s compute the shortest-paths from $u$:

**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 $S$:
    - $D(v) = \min\{D(v), D(w) + c(w,v)\}$

$D$ is initialized based on $u$’s weight, and $S$ only contains $u$ itself.

If each router knows the entire graph, then it is easy to find paths to any given destination.
\[ D(.) = \begin{array}{cccc} A & B & C & D \\ E & F & G & \infty \end{array} \]

\[ S = \{u\} \]

*Now, do it by yourself*

*Here is the final state*

*From the shortest-paths, \( u \) can directly compute its forwarding table*

*Forwarding table*

<table>
<thead>
<tr>
<th>destination</th>
<th>next-hop</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>B</td>
<td>A</td>
</tr>
<tr>
<td>C</td>
<td>E</td>
</tr>
<tr>
<td>D</td>
<td>A</td>
</tr>
<tr>
<td>E</td>
<td>E</td>
</tr>
<tr>
<td>F</td>
<td>E</td>
</tr>
<tr>
<td>G</td>
<td>E</td>
</tr>
</tbody>
</table>

*Initially, routers only know their ID and their neighbors*

D only knows, it is connected to B and C along with the weights to reach them (by configuration)
Each router builds a message (known as Link-State) and **floods it** (reliably) in the entire network.

At the end of the flooding process, everybody shares the **exact same view of the network** required for correctness. See exercise.

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
- Rely on distributed computation
- Distance-Vector
- BGP

Instead of locally computing paths based on the graph, paths can be computed in a distributed fashion:

Let $d_x(y)$ be the cost of the least-cost path known by $x$ to reach $y$.

Each node updates its distances based on neighbors' vectors:

$$d_x(y) = \min_{v} \{ c(x,v) + d_v(y) \}$$

over all neighbors $v$.

We’ll see in a few weeks how OSPF implements all this in real networks (and is used within ETH’s).

Essentially, there are three ways to compute valid routing state:
Let's compute the shortest-path from $u$ to $D$

To unfold the recursion, let's start with the direct neighbor of $D$

As soon as a distance vector changes, each node propagates it to its neighbor

As before, $u$ can directly infer its forwarding table by directing the traffic to the best neighbor the one which advertised the smallest cost

Eventually, the process converges to the shortest-path distance to each destination

Eventually, the process converges to the shortest-path distance to each destination

In few weeks, we'll learn how BGP uses distributed computation to forward packets in the Internet

The values computed by a node $u$ depends on what it learns from its neighbors ($A$ and $E$)

B and C announce their vector to their neighbors, enabling A to compute its shortest-path

Eventually, the process converges to the shortest-path distance to each destination

In few weeks, we'll learn how BGP uses distributed computation to forward packets in the Internet

As soon as a distance vector changes, each node propagates it to its neighbor

As before, $u$ can directly infer its forwarding table by directing the traffic to the best neighbor the one which advertised the smallest cost

In few weeks, we'll learn how BGP uses distributed computation to forward packets in the Internet

As soon as a distance vector changes, each node propagates it to its neighbor

As before, $u$ can directly infer its forwarding table by directing the traffic to the best neighbor the one which advertised the smallest cost

In few weeks, we'll learn how BGP uses distributed computation to forward packets in the Internet

As soon as a distance vector changes, each node propagates it to its neighbor

As before, $u$ can directly infer its forwarding table by directing the traffic to the best neighbor the one which advertised the smallest cost

In few weeks, we'll learn how BGP uses distributed computation to forward packets in the Internet

As soon as a distance vector changes, each node propagates it to its neighbor

As before, $u$ can directly infer its forwarding table by directing the traffic to the best neighbor the one which advertised the smallest cost

In few weeks, we'll learn how BGP uses distributed computation to forward packets in the Internet

As soon as a distance vector changes, each node propagates it to its neighbor

As before, $u$ can directly infer its forwarding table by directing the traffic to the best neighbor the one which advertised the smallest cost

In few weeks, we'll learn how BGP uses distributed computation to forward packets in the Internet

As soon as a distance vector changes, each node propagates it to its neighbor

As before, $u$ can directly infer its forwarding table by directing the traffic to the best neighbor the one which advertised the smallest cost

In few weeks, we'll learn how BGP uses distributed computation to forward packets in the Internet

As soon as a distance vector changes, each node propagates it to its neighbor

As before, $u$ can directly infer its forwarding table by directing the traffic to the best neighbor the one which advertised the smallest cost

In few weeks, we'll learn how BGP uses distributed computation to forward packets in the Internet

As soon as a distance vector changes, each node propagates it to its neighbor

As before, $u$ can directly infer its forwarding table by directing the traffic to the best neighbor the one which advertised the smallest cost

In few weeks, we'll learn how BGP uses distributed computation to forward packets in the Internet

As soon as a distance vector changes, each node propagates it to its neighbor

As before, $u$ can directly infer its forwarding table by directing the traffic to the best neighbor the one which advertised the smallest cost

In few weeks, we'll learn how BGP uses distributed computation to forward packets in the Internet

As soon as a distance vector changes, each node propagates it to its neighbor

As before, $u$ can directly infer its forwarding table by directing the traffic to the best neighbor the one which advertised the smallest cost

In few weeks, we'll learn how BGP uses distributed computation to forward packets in the Internet

As soon as a distance vector changes, each node propagates it to its neighbor

As before, $u$ can directly infer its forwarding table by directing the traffic to the best neighbor the one which advertised the smallest cost

In few weeks, we'll learn how BGP uses distributed computation to forward packets in the Internet

As soon as a distance vector changes, each node propagates it to its neighbor

As before, $u$ can directly infer its forwarding table by directing the traffic to the best neighbor the one which advertised the smallest cost

In few weeks, we'll learn how BGP uses distributed computation to forward packets in the Internet

As soon as a distance vector changes, each node propagates it to its neighbor

As before, $u$ can directly infer its forwarding table by directing the traffic to the best neighbor the one which advertised the smallest cost

In few weeks, we'll learn how BGP uses distributed computation to forward packets in the Internet

As soon as a distance vector changes, each node propagates it to its neighbor

As before, $u$ can directly infer its forwarding table by directing the traffic to the best neighbor the one which advertised the smallest cost

In few weeks, we'll learn how BGP uses distributed computation to forward packets in the Internet

As soon as a distance vector changes, each node propagates it to its neighbor

As before, $u$ can directly infer its forwarding table by directing the traffic to the best neighbor the one which advertised the smallest cost

In few weeks, we'll learn how BGP uses distributed computation to forward packets in the Internet

As soon as a distance vector changes, each node propagates it to its neighbor

As before, $u$ can directly infer its forwarding table by directing the traffic to the best neighbor the one which advertised the smallest cost

In few weeks, we'll learn how BGP uses distributed computation to forward packets in the Internet

As soon as a distance vector changes, each node propagates it to its neighbor

As before, $u$ can directly infer its forwarding table by directing the traffic to the best neighbor the one which advertised the smallest cost

In few weeks, we'll learn how BGP uses distributed computation to forward packets in the Internet

As soon as a distance vector changes, each node propagates it to its neighbor

As before, $u$ can directly infer its forwarding table by directing the traffic to the best neighbor the one which advertised the smallest cost

In few weeks, we'll learn how BGP uses distributed computation to forward packets in the Internet

As soon as a distance vector changes, each node propagates it to its neighbor

As before, $u$ can directly infer its forwarding table by directing the traffic to the best neighbor the one which advertised the smallest cost

In few weeks, we'll learn how BGP uses distributed computation to forward packets in the Internet

As soon as a distance vector changes, each node propagates it to its neighbor

As before, $u$ can directly infer its forwarding table by directing the traffic to the best neighbor the one which advertised the smallest cost

In few weeks, we'll learn how BGP uses distributed computation to forward packets in the Internet

As soon as a distance vector changes, each node propagates it to its neighbor

As before, $u$ can directly infer its forwarding table by directing the traffic to the best neighbor the one which advertised the smallest cost

In few weeks, we'll learn how BGP uses distributed computation to forward packets in the Internet