Communication Networks

Prof. Laurent Vanbever

Last week on Communication Networks

The Internet should allow processes on different hosts to exchange data...everything else is just commentary...

In practice, there exists a lot of network protocols. How does the Internet organize this?

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

Communication Networks

Spring 2021

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

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

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?

#5

In practice, there exists a lot of network protocols. How does the Internet organize this?
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?

We will start diving in the fundamental challenges underlying networking.

Essentially, there are three ways to compute valid routing state:

1. Use tree-like topologies - Spanning tree
2. Rely on a global network view - Link-State SDN
3. Rely on distributed computation - Distance-Vector BGP

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

We'll see how to compute spanning-trees in 2 weeks. For now, assume it is possible.

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

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

While flooding works, it is quite wasteful.
The issue is that nodes do not know their respective locations.

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 green nodes learn how to reach A.

All the nodes know on which port A can be reached.
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.

Learning is topology-dependent. The blue nodes only know how to reach A (not B).

Routing by flooding on a spanning-tree in a nutshell:
- Flood first packet to node you're trying to reach
- All switches learn where you are
- When destination answers, some switches learn where it is some because packet to you is not flooded anymore
- The decision to flood or not is done on each switch depending on who has communicated before.

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>automatically adapts to moving host</td>
</tr>
<tr>
<td>fast</td>
<td>slow to react to failures</td>
</tr>
<tr>
<td>adapts to moving host</td>
<td>host movement</td>
</tr>
</tbody>
</table>

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:
- $S = \{u\}$

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:
    - $D(v) = \min(D(v), D(w) + c(w,v))$
  - else:
    - $D(v) = \infty$
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:
    $D(v) = \infty$

$D(v)$ is the smallest distance currently known by $u$ to reach $v$.

$D(.) = \begin{array}{cccc}
A & B & C & D \\
3 & \infty & 2 & \infty \\
E & F & G & \infty \\
\infty & \infty & \infty & \infty \\
\end{array}$

Loop

while not all nodes in $S$:
  add $w$ with the smallest $D(w)$ to $S$
  update $D(w)$ for all adjacent $v$ not in $S$:
    $D(v) = \min(D(v), D(w) + c(w,v))$

$D(.) = \begin{array}{cccc}
A & B & C & D \\
3 & \infty & 2 & \infty \\
E & F & G & \infty \\
\infty & \infty & \infty & \infty \\
\end{array}$

Let $S = \{u, E\}$

$D(.) = \begin{array}{cccc}
A & B & C & D \\
3 & \infty & 2 & \infty \\
E & F & G & \infty \\
\infty & \infty & \infty & \infty \\
\end{array}$
Now, do it by yourself

This algorithm has a $O(n^2)$ complexity where $n$ is the number of nodes in the graph

From the shortest-paths, $u$ can directly compute its forwarding table

Initially, routers only know their ID and their neighbors

Each router builds a message (known as Link-State) and floods it (reliably) in the entire network

To build this global view routers essentially solve a jigsaw puzzle

Better implementations rely on a heap to find the next node to expand, bringing down the complexity to $O(n \log n)$
At the end of the flooding process, everybody share the \textit{exact same view of the network} required for correctness see exercise

Dijkstra will always converge to a unique stable state when run on \textit{static} weights cf. exercise session for the dynamic case

Essentially, there are three ways to compute valid routing state

- Use tree-like topologies
- Rely on a global network view
- Rely on distributed computation

spanning-tree
link-state
distance-vector

BGP
SDN

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

Let's compute the shortest-path from $u$ to $D$
The values computed by a node $u$ depends on what it learns from its neighbors (A and E):

$$d_u(x) = \min \{ c(x,v) + d_v(y) \}$$

over all neighbors $v$.

$$d_u(x) = \min \{ c(x,v) + d_v(y) \}$$

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

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

To unfold the recursion, let’s start with the direct neighbor of D:

$$d_u(D) = \min \{ c(u,A) + d_A(D),\ c(u,E) + d_E(D) \}$$

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

$$d_u(D) = \min \{ 3 + d_A(D),\ 2 + d_E(D) \}$$

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

$$d_u(D) = \min \{ 3 + d_A(D),\ 2 + d_E(D) \}$$

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

$$d_u(D) = \min \{ 3 + d_A(D),\ 2 + d_E(D) \}$$

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

$$d_A(D) = \min \{ 2 + d_B(D),\ 1 + d_C(D) \}$$

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

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

As before, $u$ can directly infer its forwarding table by directing the traffic to the best neighbor:

the one which advertised the smallest cost

Evaluating the complexity of DV is harder, we’ll get back to that in a couple of weeks.

Next week on Communication Networks

Reliable transport!