Computer Networks - Lecture 15 & 16

- Author: Ruben Schenk
- Date: 08.06.2021
- Contact: ruben.schenk@inf.ethz.ch

5.4.1 IPv6 Addresses

Since the IPv4 address space is running out and there is still room for different improvements there is IPv6, which features 128-bit addresses, which are denoted in 8 groups of 4 hexadecimal digits. Leading zeros are omitted and groups of zeros can be replaced by a second colon.

Example: `2001:0db8:0000:0000:0000:ff00:0042:8329` may be written as `2001:db8::ff0:42:8329`. Remember that we may only remove *one* groupe of consecutive zeros, e.g., `2001:0db8:0000:0000:1a12:0000:0000:1a13` may either be written as `2001:db8::1a12:0:0:1a13` or as `2001:db8:0:0:1a12::1a13`.

5.4.2 Tunneling

The goal of `IPv6 tunneling` is to allow IPv6 communication over IPv4. We want a *tunnel* to act as a single link across an IPv4 network. We encapsulate an IPv6 packet as payload into an IPv4 packet. The IPv6 is extracted as soo as it reaches an IPv6 link.

5.5 Routing

The goal for any `routing algorithm`, no matter which routing scheme it uses, is that it should obey the following properties:

- *Correctness*: Finds paths that work
- *Efficient paths*: The given path should be minimal for some metric
- *Fair paths*: The path doesn't starve any nodes
- *Fast convergence*: The path recovers quickly after changes
- *Scalability*: Works well as the network grows large

5.5.1 Shortest Path Routing (Dijkstra Algorithm)

To find a `shortest path` we do the following steps:

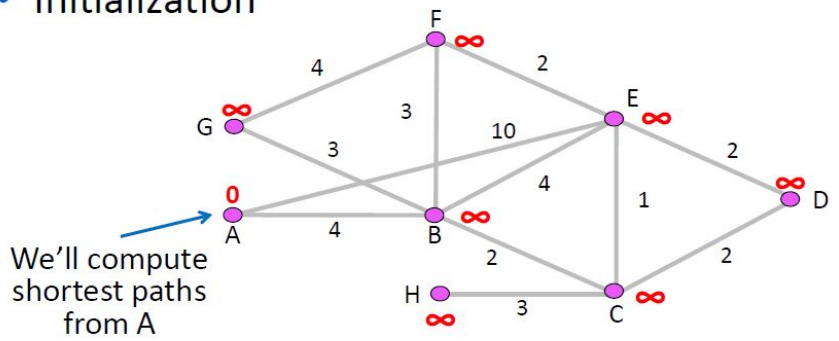
1. Assign each link a *cost* (distance).
2. Define the best path between each pair of nodes as the path that has the lowest total cost.
3. Pick randomly to break any ties.

One property when choosing the shortest path as described above, is that *sub-paths of shortest paths are also shortest paths*.

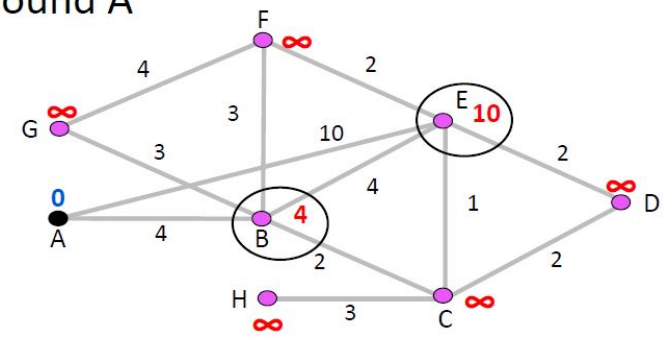
We furthermore define a `sink tree` of some node as the union of all shortest paths (i.e. the shortest path from each node) to the destination node.

The following figures show an example of how to use `Dijkstra's Algorithm`:

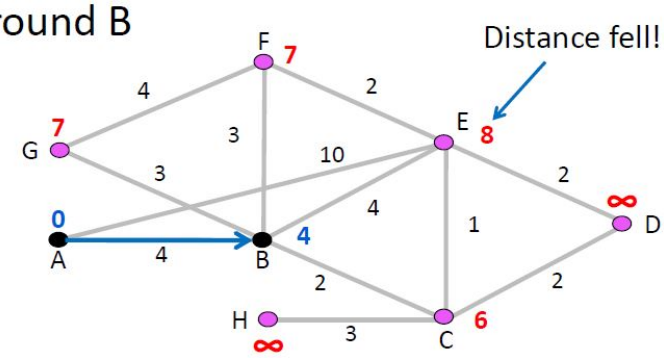
- Initialization



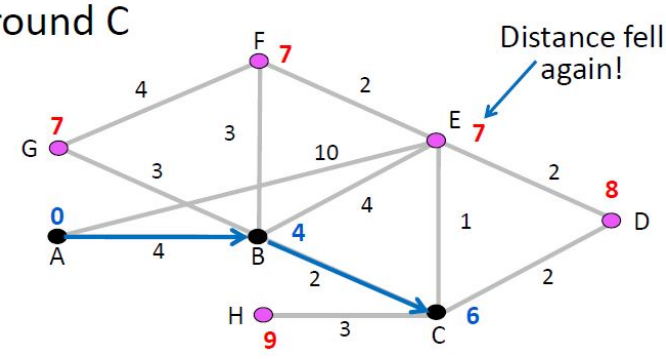
- Relax around A



- Relax around B

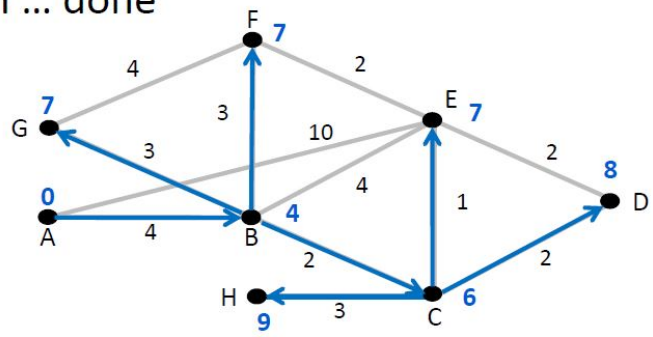


- Relax around C



(*Some intermediate steps are left out*)

- Finally, H ... done



5.5.2 Hierarchical Routing

There are several key impacts of routing growth, listed below:

- Forwarding tables grow
- Routing messages grow
- Routing computation grows

Some techniques to scale routing are:

1. IP prefixes (route to block of hosts, not individual hosts)
    - We group hosts under an IP prefix and connect them directly to the router, this way there is only one entry needed for all hosts.
2. Network hierarchy (route to network regions)
    - The idea is to introduce a larger routing unit. We then route first to the region, then to the IP prefix within the region.
3. IP prefix aggregation (combine and split prefixes)

The idea is to create subnets (*splitting*) and joining (*aggregation*) IP's based on their prefix, such that we may address a "pool" of IP's and once in there, route to the more specific IP's in the subnet. More specific:

- `Subnets`: Internally split one less specific prefix into multiple more specific prefixes.
- `Aggregation`: Externally join multiple more specific prefixes into one large prefix.

5.5.3 Distance Vector Routing

The `distance vector routing` algorithm follows a distributed BEllman-Ford approach. It works well and was used in RIP, but converges slowly after some types of failures:

- Setting: Nodes know only the cost to neighbors, not the topology. They can communicate with their neighbors via messages.
- Process: Each node maintains a `vector of distances` and next-hops to all destinations. The algorithm proceeds as follows:
    1. Initialize each vector with cost to oneself = 0 and cost to others = ∞.
    2. Periodically send this vector to the neighbors.
    3. Every round, after receiving the vectors of all neighbors:
        1. For each neighbor, add the cost of the link to the neighbor to the vector received from that neighbor.

2. Set all vector entries (except the oneself) to the minimum of all received values and set the corresponding neighbor as the next-hop.

5.5.4 Flooding

`Flooding` is used to broadcast a message to all nodes in a network in a very simple but highly inefficient way:

1. Send an incoming message to all neighbors, but
2. Remember the message (using sequence numbers) such that a message is flooded only once to the neighbors.

5.5.5 Link State Routing

The `Link State Routing Algorithm` works as follows:

1. The nodes flood the topology in the form of link state packets such that each node learns the full topology.
2. Each node computes its own forwarding table by running Dijkstra (or equivalent)

5.5.6 Distance Vector Routing vs. Link State Routing

Goal	Distance Vector	Link State
Correctness	Distributed Bellman-Ford	Replicated Dijkstra
Efficient Paths	Approximate with shortest paths	Approximate with shortest paths
Fair paths	Approximate with shortest paths	Approximate with shortest paths
Fast convergence	Slow (many exchanges)	Fast (flood and compute)
Scalability	Excellent	Moderate