Technische Universität Ilmenau 24. Februar 2024

Vorlesung

Netzalgorithmen

Prof. Dr.-Ing Günter Schäfer

NEXTEX

Zusammenfassung von
Adrian Schollmeyer

Inhaltsverzeichnis

1 Introduction

1.1 Basic Types of Transmissions

Web
- Bunch of data to be transmitted
- No guaranteed arrival times
- Simplest Case: Server and Client are directly connected by a cable

Telephony
- Continuous flow of information
- Information must arrive in time
- Simplest Case: Two telephones are directly connected via a cable

1.2 Structuring a Network

As pairwise connection of all entities with each other (thus building a complete graph of all entities) does not work, other structures need to be established. We distinguish between *end systems* (user devices) and *switching elements* (switches, routers, etc.).

End systems connect to some form of uplink to access/provide information on the network. They do not forward requests of other systems.

Switching elements Forward incoming packets onto the next hop towards its destination. The “best” next hop is decided by various routing/forwarding tables and algorithms.

1.3 Routing Algorithms

Routers execute *routing* algorithms to decide which output line an in coming packet should be transmitted on.

Connection-oriented services Run the routing algorithm during connection setup and only once to find one path to forward the packets along for the whole lifetime of the connection.

Connectionless services Run the routing algorithm either for each packet or periodically, updating the router’s forwarding table in the process.

Routing algorithms can take a *metric* into account that assigns costs to network links and allows administrators to influence routing decisions. Some possible metrics are:

- Financial cost for sending a packet over a link (e. g. when the link is charged per unit of data transferred).
- Delay (useful to penalize using a link with high delay when trying to prefer links with low delay)
- Number of hops (commonly used in most routing algorithms deployed on the Internet, aims to reduce the number of routers/networks to traverse to reach a destination)

The cheapest path is also commonly referred to as the *shortest path*.

Basic types of routing algorithms:

Non-adaptive routing algorithms do not base routing decisions on the current state of the network.

Adaptive routing algorithms take into account the current network state (e. g. distance vector routing, link state routing).

1.4 Flooding

Flooding is a simple strategy, sending every incoming packet to every outgoing link except the one it arrived on. This leads to many duplicated packets in the network, but leads to the packet almost certainly arriving at the destination (the only exception being broken links partitioning the network into two parts).

To reduce the number of duplicated packets, strategies can be used:

Solution 1: Hop counting Have a hop counter in the packet header, which is decremented by each router. If the packet stays in the network for too long, the hop counter goes to 0 and the packet is dropped. Ideally, the hop counter should be initialized to the length of the shortest path from the source to the destination.

Solution 2: Sequence numbers Each router maintains a sequence number and a table of sequence numbers it has seen from other routers. The first-hop router increments and adds its sequence number to each incoming packet from a host. Each router only forwards incoming packets, if it hasn't seen this sequence number from the first-hop router, yet. Thus, packets that have already been seen are discarded.

1.5 Adaptive Routing Algorithms

Non-adaptive routing algorithms pose problems:

- Non-adaptive routing algorithms can't cope with dramatic changes in traffic levels in different parts of the network.
- Non-adaptive routing algorithsm are usually based on average traffic conditions, but lots of computer traffic us extremely *bursty* (i. e. very variable in intensity).

Thus, adaptive routing algorithms are commonly used to make routing decisions.

Three types can be distinguished:

Centralized adaptive routing Has only one central routing controller making routing decisions.

Isolated adaptive routing Is based on information local to each router. No exchange of information between routers is required.

Distributed adaptive routing Uses periodic exchanges of information between routers to compute and update routing information to be stored in the local forwarding table.

1.5.1 Centralized Adaptive Routing

At the heart of Centralized Adaptive Routing is a central routing controller, which

- periodically collects link state information from routers
- calculates routing tables for each router
- dispatches updated routing tables to each router

The centralized approach is severely limited by the routing controller. If it goes down, the routing becomes non-adaptive, making the network vulnerable to outages. Furthermore, the controller needs to handle a great deal of routing information, making it not only a single point of failure, but also a bottleneck for scalability and performance of the network.

1.5.2 Isolated Adaptive Routing

The basic idea is to make routing decisions solely based on information available locally in each router, e. g.:

- *Hot potato*
- *Backward learning*

Hot potato routing:

- Forward the incoming packet to the output link with the shortest queue
- Do not care where the selected output link leads
- Not very effective

Backward learning:

- Maintain a local forwarding table with next hop, hop count and output link

- Incoming packets update the fowarding table entry of the sender if their hop count is better than the entry's current hop count
- Forward packets based on the forwarding table, random route (hot potato, flood) the packet if no entry for the destination exists.
- Remove/forget stale entries in the forwarding table to account for deterioration of routes (e. g. in case of link failures)

Ethernet switches commonly use backward learning to maintain forwarding tables for MAC addresses, usually falling back to flooding packets if no entry for the destination MAC exists in their local forwarding table.

1.5.3 Distributed Adaptive Routing

The central goal is to determine a "good" path (i. e. a sequence of routers) through the network from source to destination. For calculations, the network is abstracted into a graph consisting of:

Routers represented by nodes

Links represented by edges

Costs assigned to edges, representing link costs

Routing algorithms can be classified in several ways:

- Global or decentralized information

  Decentralized All routers know only a portion of the network, i. e. their physically connected neighbors and link costs to their neighbors. By exchanging information with neighbors, routes to other destinations can be calculated. Examples: BGP (path vector), RIP (distance vector)

  Global By exchanging information, routers gain knowledge of the full network topology and the cost of each link. This is used to compute cheap routes to each destination in the network. Examples: OSPF, IS-IS

- Static or dynamic routes

  Static Routes change only slowly over time, e. g. if they are statically configured by network administrator.

  Dynamic Routes change more quickly in response to link cost changes and require periodic updates of routing information.

1.5.4 Graph Model for Routing Algorithms

- $V = \{v_1, v_2, \dots, v_n\}$ the set of nodes (routers)
- $E = \{e_1, e_2, \dots, e_m\} \subseteq V^2$ the set of edges (links)
- $c : V \times V \to \mathbb{Z}_{>0}$ cost of an edge
- $s \in V$ start node (i. e. the node for which the shortest path shall be found)
- $d[i]$ cost from s to node v_i
- $p[i]$ index j of the predecessor v_j of v_i on the shortest path from s to v_i.
- $\delta(s, v)$ cost of the shortest path from s to v

1.5.5 Dijkstra's Algorithm for Shortest Paths

Maintain a set N (initially empty) of nodes for which shortest have been found. For each node i that is not directly connected to s, set $d[i] = \infty$, otherwise $d[i] = c(s, v_i)$. Starting from s, take $v_i \in V \setminus N$ with the lowest $d[i]$. The path from v_i's predecessor to v_i is the shortest path. Update $d[j]$ for neighbors $v_j \in V \setminus N$ of v_i, wherever $d[i] + c(v_i, v_j) < d[j]$ (i. e. the path from s to v_j via v_i is shorter than the previously known path from s to v_j). Set $N := N \cup \{v_i\}$. This results in finding the shortest paths from s to each node in the network.

Complexity:

- $\mathcal{O}(|V|^2)$
- Optimal in dense graphs with $|E| \approx |V|^2$
- Efficient implementations with $\mathcal{O}(|V| \cdot \log|V| + |E|)$ are possible when using Fibonacci-Heaps.

1.5.6 Distance Vector Routing

For distance vector routing, each node has its own table for $D^X(Y, Z)$, listing the cost from X to Y via Z as next hop. Distance vector routing has a few favorable properties, useful in large networks like the Internet:

Iterative The algorithm works iteratively, until no nodes exchange information (i. e. no updated information is transmitted through the network)

Asynchronous Information does not need to be exchanged in lock step. Instead any node can send updated information at any time.

Distributed Each node only needs to communicate with its directly attached neighbors.

Initially, all routers only know the costs to their neighbors and set the cost to any other destination to ∞ (aka. unreachable). Afterwards they notify neighbors of their costs to each destination they know of. Then, distance vector routing algorithms continuously wait for changes in local link costs or link update messages from neighbors. Whenever such a change occurs, the information is used to update the local distance table. If any changes to shortest (!) paths have occured, neighbors are notified of the updated shortest path costs.

The main problem of distance vector routing is the *count to infinity* problem. If the link cost for a link suddenly increases (possibly to ∞), the network might now have a shortest path to a destination going in a circle for some time, while the change in link cost is continuously increased in the network until the new cost for the link is reached or an alternative, shorter path is found. This increases the time to find a new shortest path dramatically! This issue can be mitigated with *poisoned reverse*, i. e. when Z routes to X via Z and Y notifies Z that the cost to X changed, Z notifies Y that its cost to X is ∞ to prevent Y from routing to X via Z (as Z would want to route to X via Y, making the packets go in a loop).

1.5.7 The Bellman-Ford Algorithm

The *Bellman-Ford algorithm* is capable of solving the problem of computing shortest path in graphs with edges with negative costs and is the basis for distance-vector routing. Its only limitation is that there must be no cycles with negative total cost, as otherwise the shortest path's cost will go to $-\infty$ (as continuously traversing such a cycle will inifinitely reduce the total cost). The algorithm can detect if such cycles with negative total cost exist.

The algorithm iteratively improves the estimated cost to reach a node by iterating $|V|-1$ times over all edges and check if the current estimate of the node can be improved by taking any of the connected edges, given the current estimate cost. As this algorithm always checks all edges, it has a higher running time of $\mathcal{O}\big(|V| \cdot |E|\big)$.

Negative cost cycles can be detected by running the algorithm $|V|$ times. If the cost to any node has changed in the $|V|$th iteration, there must be a negative cost cycle.

1.5.8 Comparison of Link State and Distance Vector Algorithms

Message complexity How many messages are exchanged?

- Link State: with n nodes, E links, $\mathcal{O}(n \cdot E)$ messages are sent by each node
- Distance Vector: exchange only between neighbors, but variable number of messages and variable convergence time

Speed of Convergence How long does it take until the routing table doesn't change after a link state change has occured?

- Link State: $\mathcal{O}\left(n^2\right)$ algorithm requires $\mathcal{O}(n \cdot E)$ messages
- Distance Vector: variable convergence time, in part caused by routing loops and the count-to-infinity problem

Robustness What happens if a router malfunctions?

- Link State:
  - Node can advertise incorrect link cost
  - Invalid routing table calculations only affect the malfunctioning router
- Distance Vector:
  - Nodes can advertise incorrect path costs
  - Each node's table is used (in part) by other routers, so errors can propagate through the network

1.6 Hierarchical Routing and Interconnected Networks

So far, an idealized scenario with identical routers and a "flat" network was assumed. In practice, networks scale to hundreds millions of destinations, making storing detailed routing tables for all destinations in the whole network technically impossible, due to memory limitations in routers and link overloads caused by routing table exchanges between routers. Furthermore, administrative autonomy in parts of the network (like in the Internet's autonomous systems) should allow for network administrators to control the routing (especially the routing protocol in use) in their network, independent of the rest of the network.

In *interconnected networks*, data transmission usually involves multiple networks. Routing can be distinguished into two levels:

Intradomain routing inside autonomouse systems.

Interdomain routing between autonomous systems

In the Internet, interdomain routing is done using the Border Gateway Protocol (BGP), which operates on the AS level and considers every AS as one hop. For intradomain routing, each network administrator can choose their AS's interior routing protocol (e. g. OSPF, RIP, iBGP, IS-IS).

For sending traffic between ASes, *Internet Service Providers* (ISPs) have peering agreements and connections with and to each other, making a data transfer possible. Depending on the policies and available links, traffic may not be able to be sent directly from the source ISP to the destination ISP, but needs to be sent to a different transport provider network first (transit).

Each network operator has to make decisions regarding how to handle the traffic in their network, including

- allocating enough capacity of routers and links,
- choosing a routing algorithm,
- setting link costs.

This requires estimation of *traffic demand* in the network. This can be displayed as *demand volume matrix* $H\colon \{1,\dots,n\}^2 \to \mathbb{N}$, denoting the traffic demand volume between nodes v_i and v_j as $H[i,j]$, also abbreviated h_{ij} later on.

1.7 Considerations on Traffic Demand and Link Utilization

To understand constraints on maximum link utilization, a few basic facts about the natur of Internet traffic need to be recapitulated:

- Packets are delayed in every router due to store-and-forward processing and queueing.
- Traffic congestion can occur in parts of the Internet.
- Packets may be dropped if arriving at a router with full output queues.

The task of a network designer is to design the network such that delay, congestion and the probability of packet drops are minimized, while also allowing for a reasonable utilization of the network. This is complicated by the fact that traffic arrival patterns and packet sizes in the Internet are random. In order to characterize Internet traffic behavior, large scale measurements are needed to gain insights about traffic arrival distribution and packet size distribution.

Stichwortverzeichnis