Dator- & telekommunikation

*Föreläsning 4*

# Network layer

### Background

* OSI-nivå 3 = nätverkslagret
  + Här sänds allt med **paket**
* Sändare och mottagare OSI-nivå 7
* Nivå 1 = bitar
* Nivå 2 = ramar

### Two Key Network-layer functions

| **#** | **Function** | **Description** | **Analogy** |
| --- | --- | --- | --- |
| **1.** | **Routing** | **Move** packets from router’s **input** to appropriate **output** | Process of getting though single interchange |
| **2.** | **Forwarding** | Determine **route** taken by packets from **source** to **destination**   * Routing algorithms * Executed using a routing table | Process of planning trip from source to destination. |

### Datagram forwarding table

| **About** | |
| --- | --- |
| * Destination address & output link * If certain destination address, set packet in a specific output link * At the bottom of the table there is a row that handles “otherwise”, in other words if a certain destination address can’t be found on the upper rows, there is a certain output these packets are sent to. | |
| **Applications** | |
| * IP destination address is located in the header of the arriving packet * Used by routing algorithm | |



# IPv4

### IPv4 datagram format (p. 376-378)

* **Header** is everything except from the **data** field
* Biggest problems with IPv4:
  + Variabelt datafält som gör det ineffektivt när routern ska läsa av paketets innehåll som är olika från paket till paket och
  + Att adresserna är slut

| **Fields** | |
| --- | --- |
| *Name of field* | *Description* |
| **Version number** | 4 bits that specify the IP protocol version |
| **Data** | Variable data field that depends on the length of the data |
| **Options** | Allows option header to extended |
| **Source / Dest. IP** | 2^32 IP-addresses in total. These are already consumed/used. |
| … | … |
|  | |

### IP Addressing

| **Hosts and routers** | |
| --- | --- |
| Before discussing IP-addressing, it is important to know how hosts and routers are connected to the internet.   * **Interface**: Boundary/connection between host/router and physical link is called an interface | |
| **Host** | Typically has **1** **physical** **link**. |
| **Router** | Necessarily has **≥2** **physical** **links** (between itself and the interface) because its duty is to forward datagrams, in other words it needs at least **one** **input** and **one** **output** |
| **Each interface has an IP-address** | |
| * **IP-addresses** are associated with **each** **interface** * Router måste ha subnätets IP-adress för att komma åt enheterna i det. | |

### Subnät

| **Definition subnät** | | |
| --- | --- | --- |
| * Klipper man routrarna så är de näten vars terminaler fortfarande kan kommunicera internt med varandra = subnät | | |
|  | | |
| **IP-adress** | | |
| **a\*b\*c\*d/n** | | |
| **1.** | **Nätverksmask**  **(=subnätsadress)** | n = antal bitar av de totalt 32 bitarna som utgörs av nätverksbitar |
| **2.** | **Användarbitar**  **(=host-address)** | användarbitar = **32 - n**  **Minimum** antal användarbitar: 32-1 = 31  (dvs om det finns (1 eller 2 hosts i subnät)) |
|  | | |

### IP-adress: How does a host get one?

* Hard-coded by system admin in a file
* DHCP-protokoll
  + Dynamically get address from a server
  + “plug-and-play”
  + Ges av sin ISP - internet service provide
  + Hur får ISP sina IP-adresser?
    - ICANN

### ICANN

* Allocates addresses
* Manages DNS
  + Domain Name Server converts web address to IP-address, for instance “sydsvenskan.se” ⇒ IP-address
* Assigns domain names, resolves disputes

# 

# IPv6

### Background

* Initial motivation: Too few addresses with 32 bits (IPv4)
* Fixed header size (40 bytes), different from IPv4 ⇒ **Speeds** **up** forwarding and processing of datagram by router
* No fragmentation allowed
  + Man kan alltså inte dela upp i små mindre paket, vilket det gick i IPv4

### IPv6 Header

* IP-address: 128 bitar långt (2^128 adresser totalt)
  + Representeras hexadecimalt (för oss människor)

| **Fields** | |
| --- | --- |
| *Name of field* | *Description* |
| **Version number** | 4 bits that specify the IP protocol version |
| **Priority** | This field identifies **priority** among datagrams in flow |
| **Flow label** | This field identifies **datagrams** in same “flow” (flow är ett löst/dåligt definierat begrepp) |
| **Next header** | This field identifies the **protocol** to which the contents (=datafield) of this datagram will be **delivered** (example to TCP or UDP).  This field uses the same values as the **upper layer protocol** field in **IPv4**. |
| … | … |
|  | |

* Andra skillnader från IPv4
  + **Checksum**: Existerar ej i version 6
  + **Options**: Existerar, men den inkluderas EJ i headern, därav att headern kan vara fixed, bland annat
  + **ICMPv6**: Ny version av ICMP

### IPv6 Addresses

| **Addresses** | |
| --- | --- |
| * 128 bits long, written in hexadeciamal * Each hexadecimal is separated by colons (:) | |
| **Example** | 3ffe:1900:4545:3:200:f8ff:fe21:67cf |

| **Rules (simplifying an address)** | | | |
| --- | --- | --- | --- |
| **Example address:** fe80:0000:0000:0000:0200:f8ff:fe21:67cf | | | |
| ***#*** | | ***Rule desc.*** | ***Example*** |
| **1.** | | **Leading** **zeros** can be omitted in **each** **field** | ⇒ fe80:0:0:0:200:f8ff:fe21:67cf |
| **2.** | | A **double** **colon** can be used **once** in an address to replace **multiple** **fields** of **zeros**. | ⇒ fe80::200:f8ff:fe21:67cf |

### 

### Transition from IPv4 to IPv6

* Alla routrar kan EJ uppgraderas samtidigt
* Hur kommer då nätverket fungera?

| **3 ways of combining datagrams of IPv4 and IPv6 in a network** | | |
| --- | --- | --- |
| **1.** | **Tunneling** | SE NEDAN |
| **2.** | **Dual Stack** | Both IPv4 and IPv6 protocols are implemented in the routers. |
| **3.** | **Translation** | When transiting, translate between protocols (information lost) |

### 

### Tunneling

* Man tar IPv6-paketet
* Stoppar in som data i IPv4 paketet
* När IPv4 paketet når IPv6-router så extraheras datan och man erhåller hela IPv6-paketet



# Routingalgoritmer

* Bygger på grafteori



* Två sätt att hantera dessa på

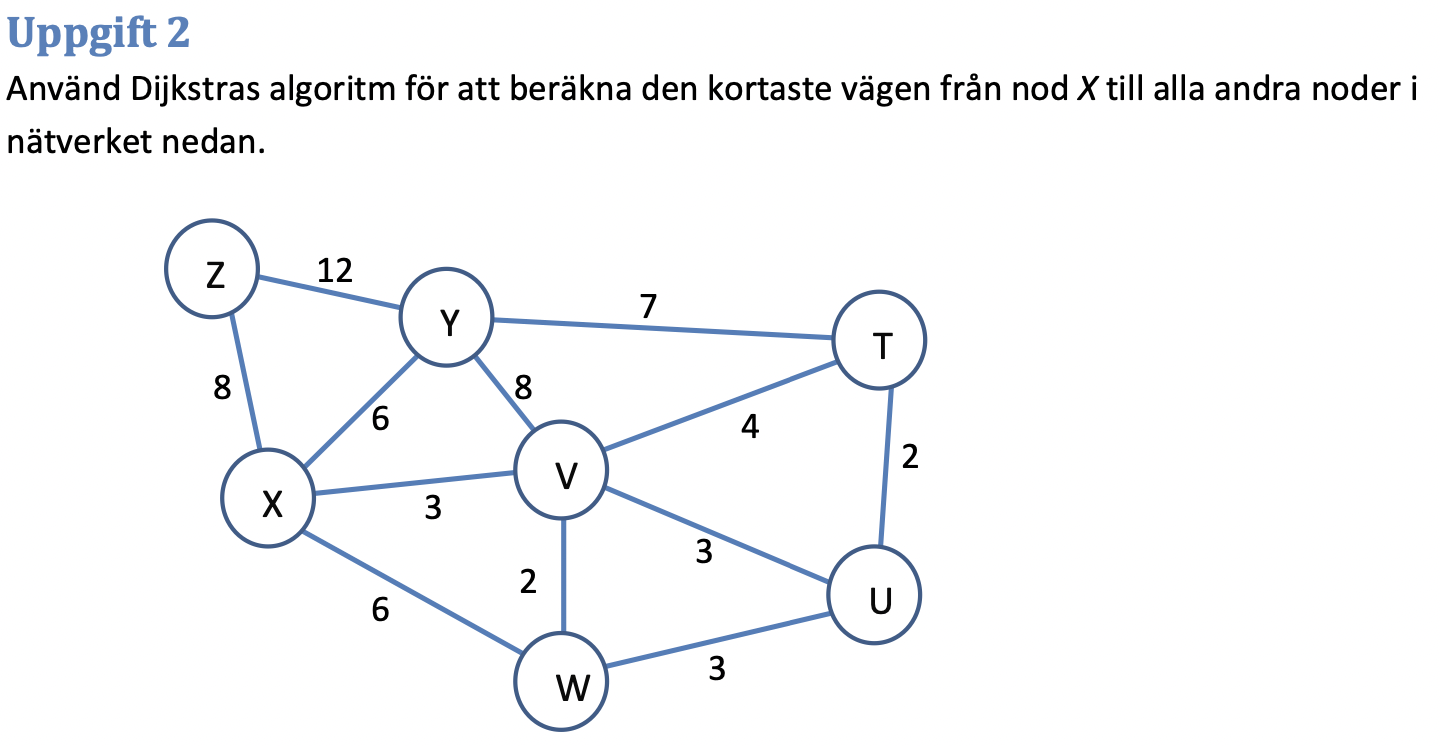
| **2 typer av routingalgoritmer** | | |
| --- | --- | --- |
| **1.** | **Link state algorithm**  (Globalt) | * Routrarna känner till alla noder och länkar (dvs all nätverkstopologi) * Kan ex. göras internt inom en verksamhet * **Dijkstra’s algorithm** |
| **2.** | **Distance vector algorithm**  (Decentraliserat) | * Router känner till fysiskt kopplade grannar * **RIP - Routing Internet Protocol** |

**Link-state-algorithm**

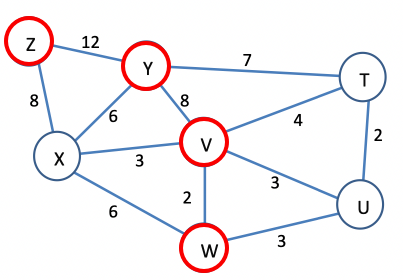
* **Dijkstra’s algorithm**
* Alla noder känner till samtliga länk-kostnader
* c(x,y): kostnad för y
* D(v); vilka noder har jag kollat på hittills
* Mängden av alla noder

### 

### Dijkstra’s Algoritm - Hur ska man tänka?



**Steg 1** - Identifiera X:s grannar.



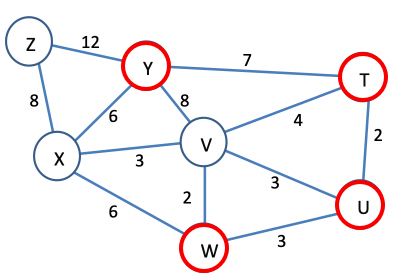
**Steg 2** - Rita upp tabellen för den **första iterationen** med;

* **N’-kolumnen** (= set of nodes whose least cost path is definitely known)
  + Vi börjar med start-noden, dvs vår utgångspunkt (X)
* **D(nod)-kolumnen** (= nuvarande kostnad för att ta sig till noden i fråga)
  + Börja med kostnaden för X:s grannar (därav steg 1, de grönmarkerade kolumnerna)
  + De noder som EJ går att nå sätts till oändligheten som kostnad
  + Vi skriver **c, Nod ⇒** Dvs **c = kostnad** ; **nod = genom vilken nod**
* Ringa avslutningsvis in den **billigaste** vägen, efter den första iterationen! (**3,X**)

|  | N’ | D(Z) | D(Y) | D(V) | D(W) | D(T) | D(U) |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | X | 8, X | 6, X | **3, X** | 6, X | inf | inf |

**Steg 3** - Fortsätt med nästa iteration

* Den billigaste vägen tar vi, så att nu blir istället N’ = XV
* Dvs nu står vi på nod V!
* Eftersom vi hittade den billigaste vägen till V, från X i första iterationen så fyller vi INTE längre in D(V)-kolumnen!
* OBS!!
  + Om vi inte kan nå en särskild nod efter den första iterationen, tar vi med den förra kostnaden
  + Om vi inte hittar en billigare kostnad än den förra, tar vi med den förra kostnaden.



|  | N’ | D(Z) | D(Y) | D(V) | D(W) | D(T) | D(U) |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | X | 8, X | 6, X | **3, X** | 6, X | inf | inf |
| 1 | **XV** | 8, X | 6, X |  | 5, V | 7, V | 6, V |

* Notera att från kostnaden till W, från XV, dvs D(W) uppdaterades eftersom vi hittade en billigare väg genom V (X → V → W ⇒ 3 + 2 = 5)

**Steg 4** - Fortsätt så här tills full tabell är uppnådd (dvs sista inrigningen)

|  | N’ | D(Z) | D(Y) | D(V) | D(W) | D(T) | D(U) |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | X | 8, X | 6, X | **3, X** | 6, X | inf | inf |
| 1 | XV | 8, X | 6, X |  | 5, V | 7, V | 6, V |
| 2 | XVW | 8, X | 6, X |  |  | 7, V | 6, V |
| 3 | XVWY | 8, X |  |  |  | 7, V | 6, V |
| 4 | XYVWYU | 8, X |  |  |  | 7, V |  |
| 5 | XYVWYU | 8, X |  |  |  |  |  |

**Distance vector algorithm**

* RIP
* Bellman-Ford Equation