CS 755 – System and Network Architectures and Implementation

Module 2 - Networks

Martin Karsten

mkarsten@uwaterloo.ca
Notice

Some slides and elements of slides are taken from third-party slide sets. In this module, parts are taken from the Kurose/Ross slide set. See detailed statement on next slide...
A note on the use of these ppt slides:
We're making these slides freely available to all (faculty, students, readers). They're in PowerPoint form so you can add, modify, and delete slides (including this one) and slide content to suit your needs. They obviously represent a lot of work on our part. In return for use, we only ask the following:
- If you use these slides (e.g., in a class) in substantially unaltered form, that you mention their source (after all, we'd like people to use our book!)
- If you post any slides in substantially unaltered form on a www site, that you note that they are adapted from (or perhaps identical to) our slides, and note our copyright of this material.

Thanks and enjoy! JFK/KWR

All material copyright 1996-2009
J.F Kurose and K.W. Ross, All Rights Reserved

Computer Networking: A Top Down Approach
5th edition.
Jim Kurose, Keith Ross
Addison-Wesley, April 2009.
Overview

• graph abstraction
• routing and forwarding
• scalability: hierarchy and aggregation
• virtualization
Channels – Review

• multiple stations share channel
• assumption: every station can reach all others
  – not entirely true for radio channels...
• main concerns
  • transmission of meaningful units, error control
  • medium access control
• labelling? yes, for filtering (not reachability)
  • add sender/receiver labels to message
    – not strictly needed for point-to-point links
Network

• consider network as partially connected graph

• no direct reachibility between all pairs of nodes
Node Labelling

• assign global label to each node – *address*
• compare with postal address
  • hierarchical
    – uniqueness
  • location-dependent
    – implicitly hierarchical

• network address – may or may not be hierarchical or location-dependent
Interface Labelling

- assign label to each interface at each node
  - global vs. local ('eth0')
- with node labelling
  - need/want at least neighbour-to-interface mapping
  - e.g. at Node U (#Slide 6)
    - V -> eth0
    - W -> eth1
    - X -> eth2
Terminology

• communication session
  • *unicast* (1-to-1) – in focus here
  • *multicast* (1-to-many)
  • *broadcast* (1-to-all) – what does 'all' mean? (*scope*)

• end system: *host*
  • *sender* or *source*
  • *receiver* or *destination*
Terminology (cont'd)

- intermediate system: *router*
  - vs. *hub* vs. *switch* – details later
- *routing*
  - dissemination of topology information
  - *path computation*
- *forwarding*
  - *path selection*
  - move messages from input link to output link
Return Path Announcement

- assume forward path exists and is used
- assume symmetric return path
- record return path
  - in message
  - in routers (vs. switch)

- assume previously shown graph (#Slide 6)
Source Routing

- message from U to Z travels via
  - U/eth2 -> X/eth0
  - X/eth3 -> W/eth0
  - W/eth1-> Z/eth1
- record eth0, eth0, eth1 -> can reverse and use!
Source Routing (alt version)

- assume globally unique names
- message from U to Z travels via X and W
  - record path in message: U, X, W, Z
  - reverse path at receiver: Z, W, X, U
- use reverse path to reach U from Z
- use local neighbour table to find interface
Self Learning

• message from U to Z travels via X and W
• assume neighbour table, then
  • record at W: U -> X
  • record at Z: U -> W
• can send message to Z 'directly'
  • without including path
  • at each router: look up table entries
Bootstrapping

- first message announces reachability

- transmission of first message?
  - broadcast – e.g., Ethernet switching
  - unicast – using some other method
Switched Ethernet

- globally unique MAC addresses
  - admin hierarchy through IEEE
- switch records information from arriving frame
  - store source address -> interface in switch table
- switch looks up destination address
  - found -> forward via interface
  - not found -> broadcast to all interfaces
Ethernet – Hierarchical Topology

- works just fine

- self-learning algorithm adapts automatically

- but: broadcast overhead?
Ethernet – History

- initial version: bus/cable
  - signal transmission limitations
  - cabling structure? cable break?
- next version: star topology
  - repeater – extend signal reach
  - hub (multiple interfaces) – permit structured cabling
- current version: switched
  - reduce broadcast effects / isolate collision domains
  - intelligence: self-learning & buffering
Virtual Circuit

- similar to self-learning: return path announcement
- use local labels, instead of addresses (#Slide 6)
  - at U: store a -> application, announce U/a
  - at X: store b -> U/a, announce X/b
  - at W: store c -> X/b, announce W/c
  - at Z: return label is W/c
- need neighbour tables (or use interface labels)
  - forwarding: replace label and forward message
Virtual Circuit

• rationale
  • can set up circuit per session (management)
  • number of sessions $\ll$ number of end systems
  • use (and reuse) limited range of local labels

=> compact table, fast lookup

• array vs. tree

• but: circuit management overhead

• home exercise – verify: NAT uses VC principles
Routing

- asynchronous topology discovery
  - decoupled from message transfer
- goals
  - discover available paths and characteristics
  - choose between paths
    - lowest cost, best service
    - get rid of packet asap
    - do not send via provider X
    - but also: maintain system consistency and stability
Graph with Link Costs

- cost: money, delay, load, etc.
- algorithms: cost must be positive and additive
Dijkstra's Algorithm

- global information: cost of all links in network
- Notation (at one node)
  - $c(x,y)$: link cost from node $x$ to $y$
  - $D(v)$: current cost of path to $v$
  - $p(v)$: last predecessor on path to $v$
  - $N'$: set of nodes whose least cost path is known
- iterative algorithm:
  after $k$ iterations, algorithm has computed $k$
  least-cost paths to $k$ nearest destinations
Dijkstra's Algorithm

1  *Initialization:*
2    \[ N' = \{u\} \]
3    for all nodes \( v \)
4      if \( v \) adjacent to \( u \)
5        then \( D(v) = c(u,v) \)
6      else \( D(v) = \infty \)
7  *Loop*
8    find \( w \) not in \( N' \) such that \( D(w) \) is a minimum
9    add \( w \) to \( N' \)
10   update \( D(v) \) for all \( v \) adjacent to \( w \) and not in \( N' \) : 
11      \[ D(v) = \min( D(v), D(w) + c(w,v) ) \]
12      /* new cost to \( v \) is either old cost to \( v \) or known 
13         shortest path cost to \( w \) plus cost from \( w \) to \( v \) */
14   *until all nodes in \( N' \)*
# Dijkstra's Algorithm – Example

<table>
<thead>
<tr>
<th>Step</th>
<th>N'</th>
<th>D(v),p(v)</th>
<th>D(w),p(w)</th>
<th>D(x),p(x)</th>
<th>D(y),p(y)</th>
<th>D(z),p(z)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>u</td>
<td>2,u</td>
<td>5,u</td>
<td>1,u</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>1</td>
<td>ux</td>
<td>2,u</td>
<td>4,x</td>
<td>2,x</td>
<td>4,y</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>uxy</td>
<td>2,u</td>
<td>3,y</td>
<td>4,y</td>
<td>4,y</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>uxyv</td>
<td>2,u</td>
<td>3,y</td>
<td>4,y</td>
<td>4,y</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>uxyvw</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>uxyvwz</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

![Graph Diagram]
Link State Routing

- routing protocols, e.g., OSPF
- establish scope, then disseminate link information “globally”
- update periodically and when link changes
- run Dijkstra's algorithm at each router
  - convergence phase during updates
- $O(n^2)$ runtime, broadcast updates, scalability?
Distance Vector Algorithm

- local information:
  - cost of links to all neighbours
  - neighbours' current costs to all known destinations
- Notation
  - \( c(x,y) \): link cost from node \( x \) to \( y \)
  - \( d(x,y) \): cost of known least-cost path from \( x \) to \( y \)
- Then: \( d(x,y) = \min_v \{ c(x,v) + d(v,y) \} \)
  - repeated iterative application converges to least-cost of paths and known next hop (Bellman-Ford)
D_x(y) = \min\{c(x,y) + D_y(y), c(x,z) + D_z(y)\} \\
= \min\{2+0, 7+1\} = 2 \\

D_x(z) = \min\{c(x,y) + D_y(z), c(x,z) + D_z(z)\} \\
= \min\{2+1, 7+0\} = 3 

**node x table**

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>0</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>y</td>
<td>∞</td>
<td></td>
<td>∞</td>
</tr>
<tr>
<td>z</td>
<td>∞</td>
<td></td>
<td>∞</td>
</tr>
</tbody>
</table>

**node y table**

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>y</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>z</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
</tbody>
</table>

**node z table**

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>y</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>z</td>
<td>7</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
\[ D_x(y) = \min\{c(x,y) + D_y(y), c(x,z) + D_z(y)\} = \min\{2+0, 7+1\} = 2 \]

\[ D_x(z) = \min\{c(x,y) + D_y(z), c(x,z) + D_z(z)\} = \min\{2+1, 7+0\} = 3 \]
Distance Vector – Challenges

- link cost changes
  - node detects local change
  - updates local table
  - if necessary, send updates
- “good news travels fast”
- “bad news travels slow”
- “count to infinity” problem
Distance Vector – Options

- convergence time during updates might be long
  - transient routing loops are problematic
- approaches
  - poisoned reverse: don't send route to next hop
    - only avoids small 3-hop loops
  - path vector: keep and transmit full path
    - avoids loops, but overhead and transparency?
- synchronous updates -> see literature
Distance Vector Routing

- routing protocols, e.g., RIP, BGP
- disseminate local routing table to neighbours
- update periodically and when table changes
- update local table at each router
  - convergence phase during updates
- $O(n)$ runtime, local updates
- potentially slow convergence, transient loops
Characteristics for Comparison

- message overhead
  - message number vs. transmission scope
- computational overhead
  - vs. frequency of updates
- robustness
  - impact of failures
- policy support
  - transparency might be a good or a bad thing
Other Aspects

- adjust routing dynamically to load changes?
  - might be unstable

- policy routing, BGP local preference
  - might result in inconsistent routing

- route information called *advertisement*
  - advertise reachability via gateway
  - somewhat similar to return path announcement
Scalability

- destination-based routing and forwarding vs. billions of nodes?
  => hierarchical addressing and routing
    - administrative autonomy for networks
    - business relationships between networks
- Internet = network of networks

- terminology: *autonomous system* (AS)
  - network – administrative unit
Hierarchical Routing

- interconnected ASes
Hierarchical Routing

• suppose X reachable from AS1 via AS2 or AS3
• configure forwarding table in router 1d
  • inter-domain routing
  • local (cost between routers) vs. global (cost between AS)es concerns?
Hierarchical Addressing

- assign contiguous addresses to subnets
  - identified by address prefix
- portion of provider's address space
- provider advertises aggregated prefix

<table>
<thead>
<tr>
<th>ISP's block</th>
<th>Organization 0</th>
<th>Organization 1</th>
<th>Organization 2</th>
<th>Organization 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>11001000 00010111 00010000 00000000</td>
<td>11001000 00010111 00010000 00000000</td>
<td>11001000 00010111 00010010 00000000</td>
<td>11001000 00010111 00011110 00000000</td>
<td></td>
</tr>
<tr>
<td>200.23.16.0/20</td>
<td>200.23.16.0/23</td>
<td>200.23.18.0/23</td>
<td>200.23.30.0/23</td>
<td></td>
</tr>
</tbody>
</table>
Hierarchical Addressing

- Organization 0
  - 200.23.16.0/23
- Organization 1
  - 200.23.18.0/23
- Organization 2
  - 200.23.20.0/23
- Organization 7
  - 200.23.30.0/23

Fly-By-Night-ISP

- “Send me anything with addresses beginning 200.23.16.0/20”
- “Send me anything with addresses beginning 199.31.0.0/16”

ISPs-R-Us

- Fundamentally: tree vs. graph
Hierarchical Addressing

- deaggregation when network moves
- also: multi-homing

```
Send me anything with addresses beginning 200.23.16.0/20
```

```
Send me anything with addresses beginning 199.31.0.0/16 or 200.23.18.0/23
```
Location and Addressing

- location-independent MAC addresses
  - hard-coded in firmware, globally unique
  - Ethernet self-learning algorithm: plug-and-play
  - scaling limitations

- topological IP addresses
  - configured, must be globally unique for responders
    - otherwise NAT is an option
  - scalable, but: network is more densely connected
  - use graph features (redundancy) -> deaggregation
Initiator vs. Responder

- who needs globally routable address?
  - initiator: party to initiate conversation
  - responder: party that accepts conversations
- only responders need globally routable address
  - e.g., initiators work well before NAT
  - service directory (e.g., VoIP)
    => maintain initiator role for responder functionality
  - service directory itself is responder
- is your laptop a responder?
Other Protocols

• Address Resolution Protocol (ARP)
  • request MAC address using broadcast
  • “who knows 10.2.57.10?” -> that node responds
  • broadcast overlaps nicely with Eth self-learning

• Dynamic Host Configuration Protocol (DHCP)
  • server manages pool of IP addresses
  • station asks for IP address during bootstrap
  • MAC broadcast request -> server responds
  • broadcast response -> coordinate multiple servers
Virtualization

- build virtual network graphs on top of networks
- use encapsulation and layering
- examples
  - IP over Ethernet
  - Virtual LANs
  - IP over IP
  - etc...
Virtual LAN (VLAN)

- what if CS user moves office to EE floor?
- single broadcast domain (ARP, DHCP) – security/privacy?
- switches not well utilized
VLAN

- switch can be configured to define multiple virtual LANs over single physical infrastructure

Port-based VLAN: switch ports grouped (by switch management software) so that single physical switch …..

... operates as multiple virtual switches

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

Electrical Engineering (VLAN ports 1-8)  Computer Science (VLAN ports 9-15)
Port-based VLAN

- traffic isolation: broadcast restricted to VLAN
- membership: based on port or MAC address
- forwarding between VLANs: routing
Multi-Switch VLAN

- trunk port: connect switches
- frames forwarded between switches must carry VLAN identifies -> extended protocol format
- IEEE 802.1q defines extra header fields
IP over IP

- example: IPV6 over IPv4
- take arbitrary subset of connected IPv4 nodes
- add IPv6 capability to those nodes
- treat IPv4 as virtual links between IPv6 nodes
  => virtual network

- IP was designed to form overlay network
**IP Tunneling**

**Logical view:**
- A: IPv6
- B: IPv6
- E: IPv6
- F: IPv6
- Tunnel

**Physical view:**
- A: IPv6
- B: IPv6
- C: IPv4
- D: IPv4
- E: IPv6
- F: IPv6

Flow: X
- Src: A
- Dest: F
- Data

A-to-B: IPv6

B-to-C: IPv6 inside IPv4

B-to-C: IPv6 inside IPv4

E-to-F: IPv6

Flow: X
- Src: A
- Dest: F
- Data