Chapter 8
LOCATION SERVICES

Distributed Computing Group

Mobile Computing

Winter 2005 / 2006

Overview

- **Mobile IP**
  - Motivation
  - Data transfer
  - Encapsulation

- **Location Services & Routing**
  - Classification of location services
  - Home based
  - GLS
  - MLS

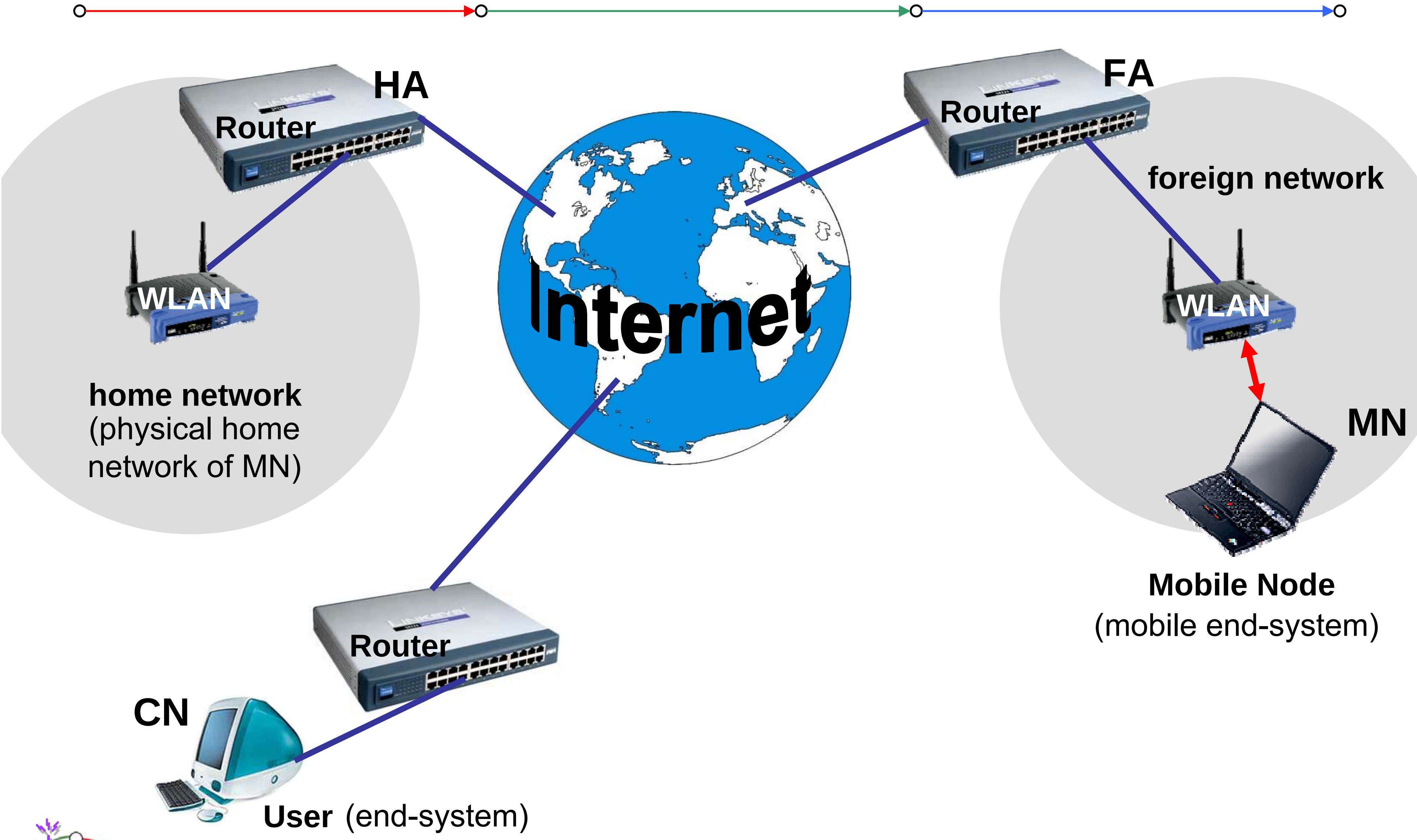
Motivation for Mobile IP

- **Routing**
  - based on IP destination address, network prefix (e.g. 129.132.13) determines physical subnet
  - change of physical subnet implies change of IP address to have a topological correct address (standard IP) or needs special entries in the routing tables
- **Changing the IP-address?**
  - adjust the host IP address depending on the current location
  - almost impossible to find a mobile system, DNS updates are too slow
  - TCP connections break
  - security problems
- **Change/Add routing table entries for mobile hosts?**
  - worldwide!
  - does not scale with the number of mobile hosts and frequent changes in their location

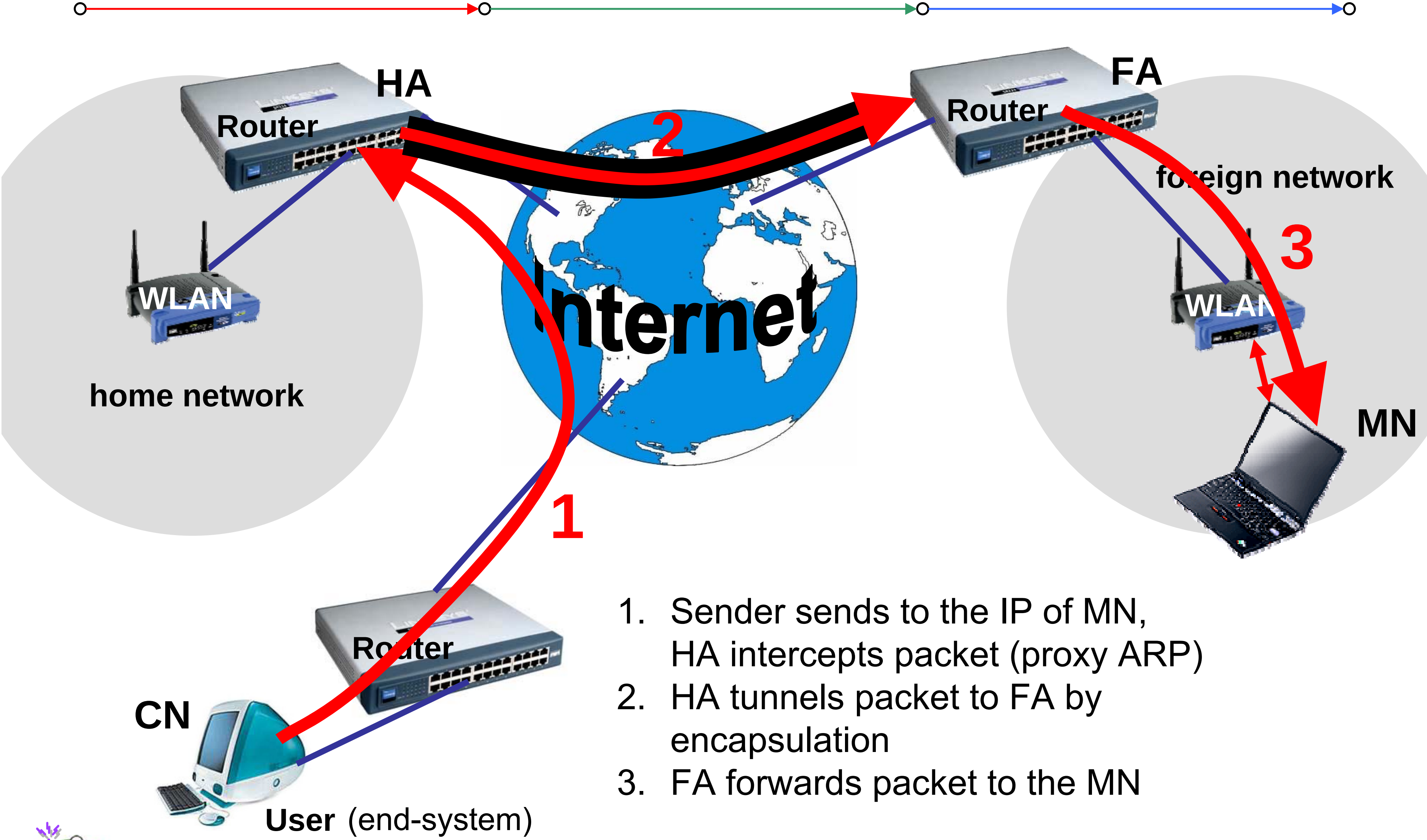
Requirements on Mobile IP (RFC 2002)

- **Compatibility**
  - support of the same layer 2 protocols as IP
  - no changes to current end-systems and routers required
  - mobile end-systems can communicate with fixed systems
- **Transparency**
  - mobile end-systems keep their IP address
  - continuation of communication after interruption of link possible
  - point of connection to the fixed network can be changed
- **Efficiency and scalability**
  - only little additional messages to the mobile system required (connection typically via a low bandwidth radio link)
  - world-wide support of a large number of mobile systems
- **Security**
  - authentication of all registration messages

Data transfer from mobile system

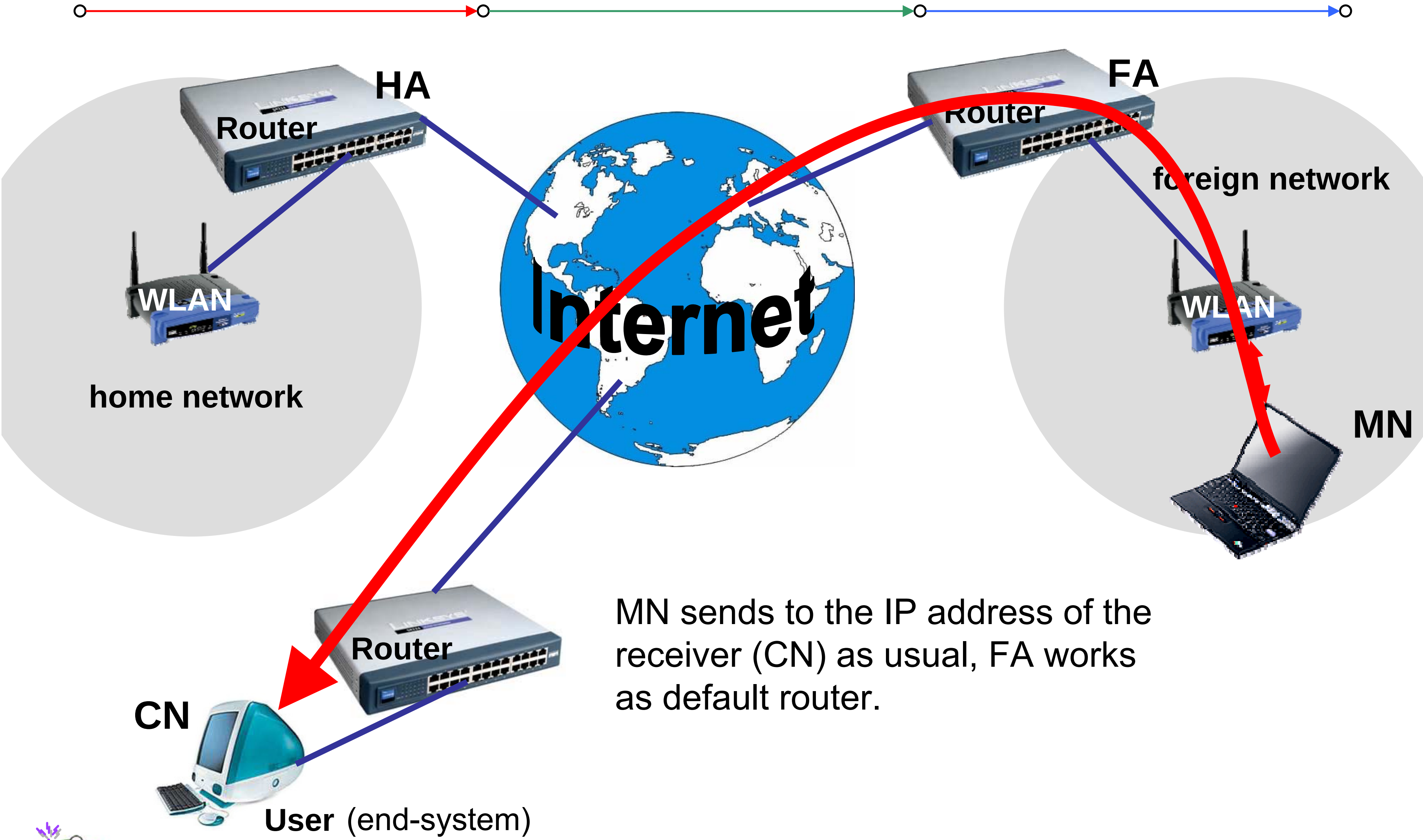


Data transfer to mobile system



1. Sender sends to the IP of MN, HA intercepts packet (proxy ARP)
2. HA tunnels packet to FA by encapsulation
3. FA forwards packet to the MN

Data transfer back to CN

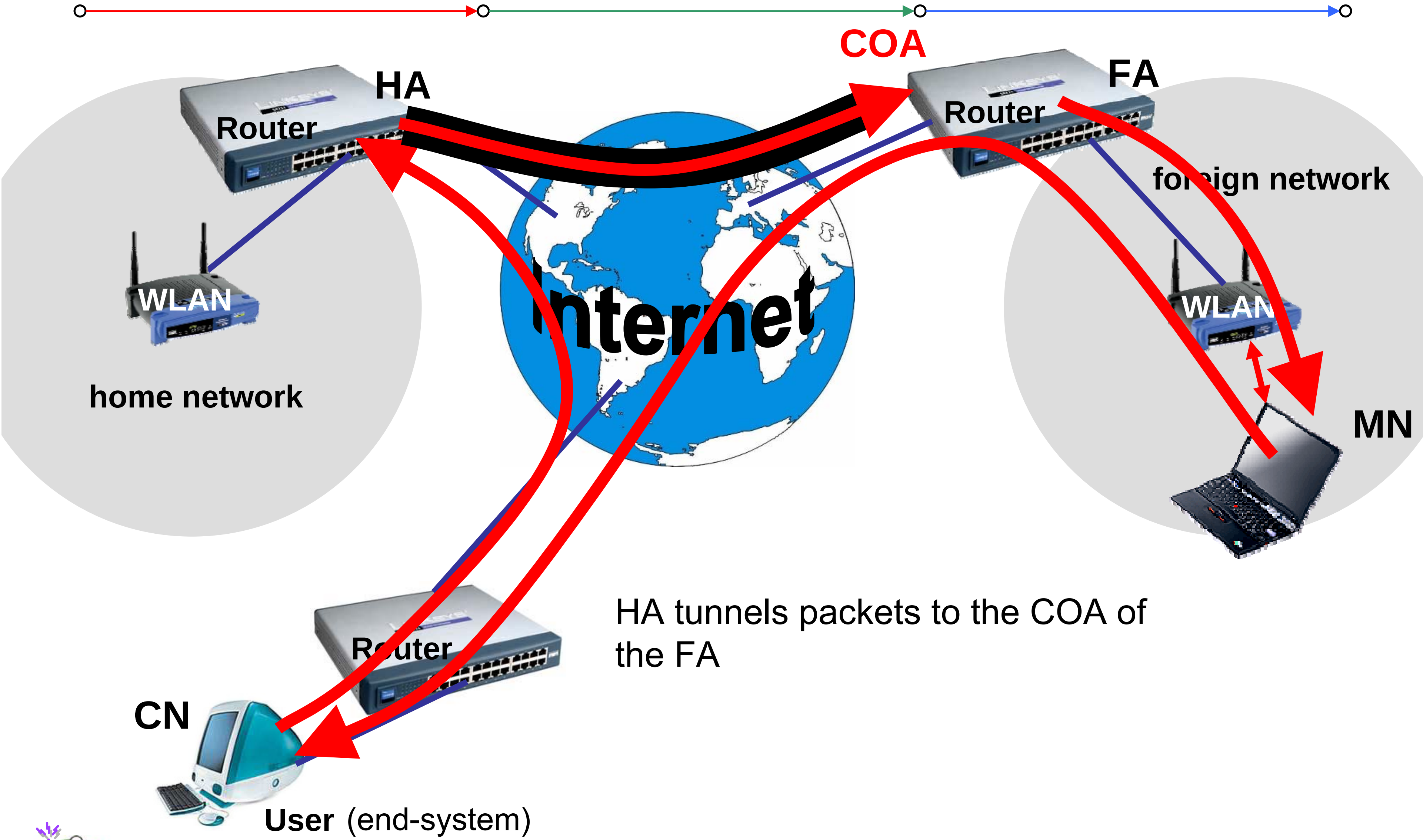


MN sends to the IP address of the receiver (CN) as usual, FA works as default router.

Terminology

- **Mobile Node (MN)**
  - system (node) that can change the point of connection to the network without changing its IP address
- **Home Agent (HA)**
  - system in the home network of the MN, typically a router
  - registers the location of the MN, tunnels IP datagrams to the COA
- **Foreign Agent (FA)**
  - system in the current foreign network of the MN, typically a router
  - typically the default router for the MN
- **Care-of Address (COA)**
  - address of the current tunnel end-point for the MN (at FA or MN)
  - actual location of the MN from an IP point of view
  - can be chosen, e.g., via DHCP
- **Correspondent Node (CN)**

Overview



HA tunnels packets to the COA of the FA

How it works…

- **Agent Advertisement**
  - HA and FA periodically send advertisement messages into their physical subnets
  - MN listens to these messages and detects if it is in the home or a foreign network (standard case for home network)
  - MN reads a COA from the FA advertisement messages
- **Registration (always limited lifetime!)**
  - MN signals COA to the HA via the FA, HA acknowledges via FA to MN
  - these actions have to be secured by authentication
- **Advertisement**
  - HA advertises the IP address of the MN (as for fixed systems), i.e. standard routing information
  - routers adjust their entries, these are stable for a longer time (HA responsible for a MN over a longer period of time)
  - packets to the MN are sent to the HA
  - independent of changes in COA/FA

Agent advertisement

0 … 7	8 … 15	16 … 23	24 … 31
type	code	checksum	
#addresses	addr. size	lifetime	
router address 1			
preference level 1			
router address 2			
preference level 2			

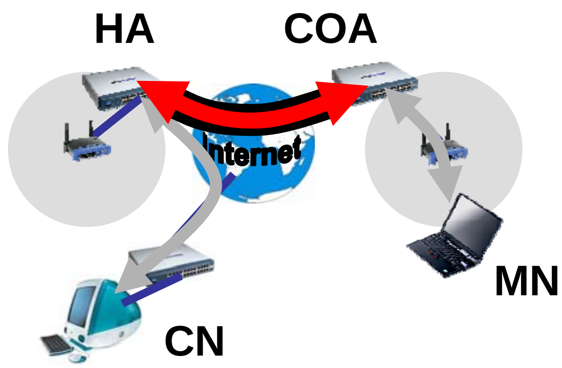
. . .

type	length	sequence number	
registration lifetime		R B H F M G V	reserved
COA 1			
COA 2			

. . .

IP-in-IP Encapsulation

- Mandatory in RFC 2003
- tunnel between HA and COA



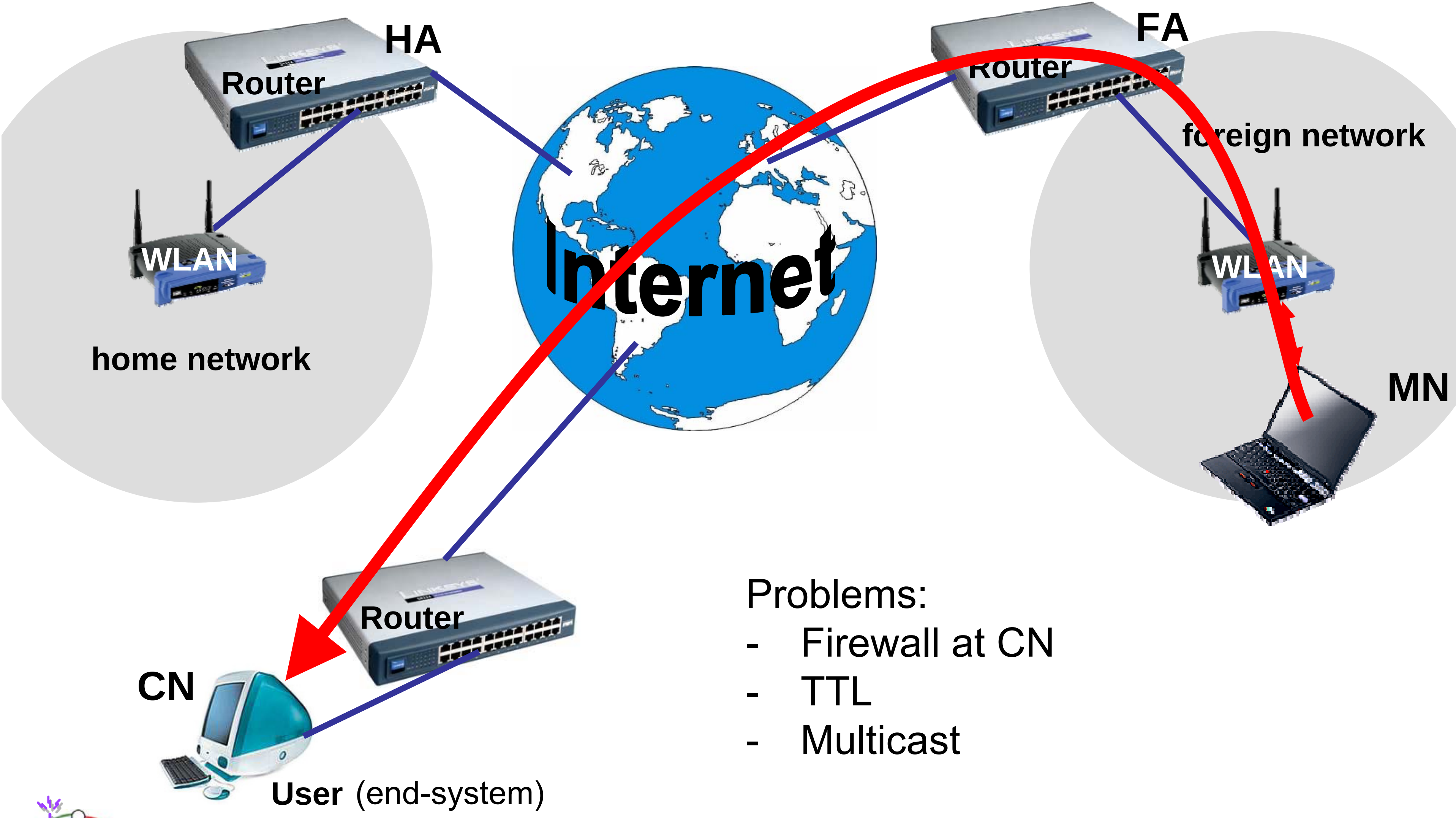
<table>
<tr><td>ver.</td><td>IHL</td><td>TOS</td><td colspan="2">length</td></tr>
<tr><td colspan="3">IP identification</td><td>flags</td><td>fragment offset</td></tr>
<tr><td colspan="2">TTL</td><td>IP-in-IP</td><td colspan="2">IP checksum</td></tr>
<tr><td colspan="5">IP address of HA</td></tr>
<tr><td colspan="5">Care-of address COA</td></tr>
<tr><td>ver.</td><td>IHL</td><td>TOS</td><td colspan="2">length</td></tr>
<tr><td colspan="3">IP identification</td><td>flags</td><td>fragment offset</td></tr>
<tr><td colspan="2">TTL</td><td>lay. 4 prot.</td><td colspan="2">IP checksum</td></tr>
<tr><td colspan="5">IP address of CN</td></tr>
<tr><td colspan="5">IP address of MN</td></tr>
<tr><td colspan="5">TCP/UDP/ ... payload</td></tr>
</table>

Minimal Encapsulation

- optional
- avoids repetition of identical fields such as TTL, IHL, version, TOS
- only applicable for unfragmented packets, no space left for fragment identification

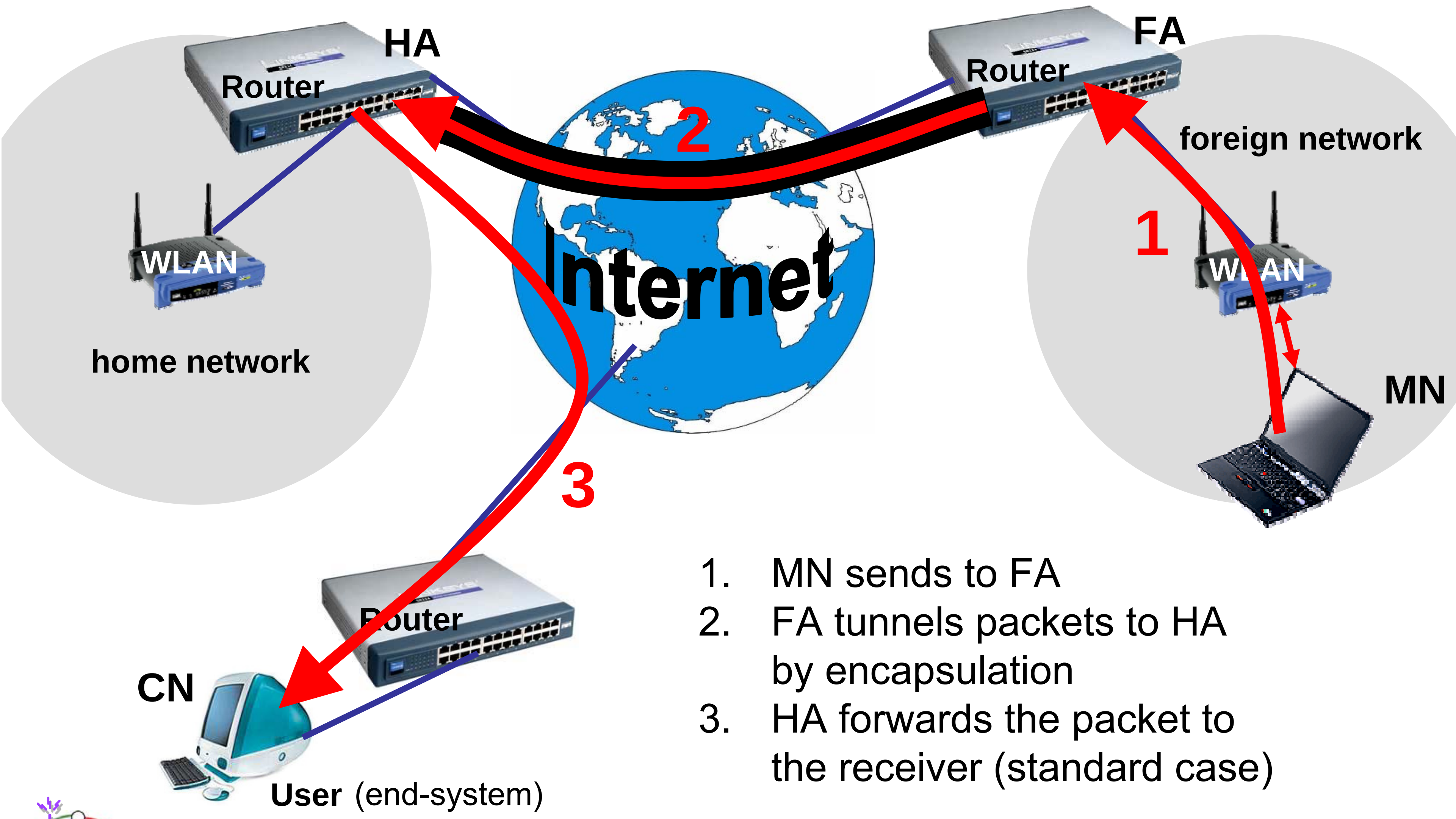
<table>
<tr><td>ver.</td><td>IHL</td><td colspan="2">TOS</td><td colspan="2">length</td></tr>
<tr><td colspan="4">IP identification</td><td>flags</td><td>fragment offset</td></tr>
<tr><td colspan="2">TTL</td><td colspan="2">min. encap.</td><td colspan="2">IP checksum</td></tr>
<tr><td colspan="6">IP address of HA</td></tr>
<tr><td colspan="6">care-of address COA</td></tr>
<tr><td colspan="2">lay. 4 protoc.</td><td>S</td><td>reserved</td><td colspan="2">IP checksum</td></tr>
<tr><td colspan="6">IP address of MN</td></tr>
<tr><td colspan="6">IP address of CN (only if S=1)</td></tr>
<tr><td colspan="6">TCP/UDP/ ... payload</td></tr>
</table>

Data transfer from the mobile system



Problems:
- Firewall at CN
- TTL
- Multicast

Reverse tunneling (RFC 2344)

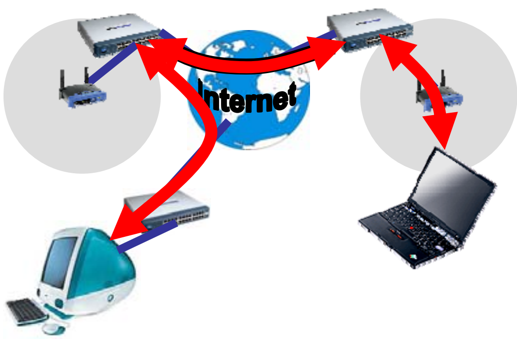


1. MN sends to FA
2. FA tunnels packets to HA by encapsulation
3. HA forwards the packet to the receiver (standard case)

Mobile IP with reverse tunneling

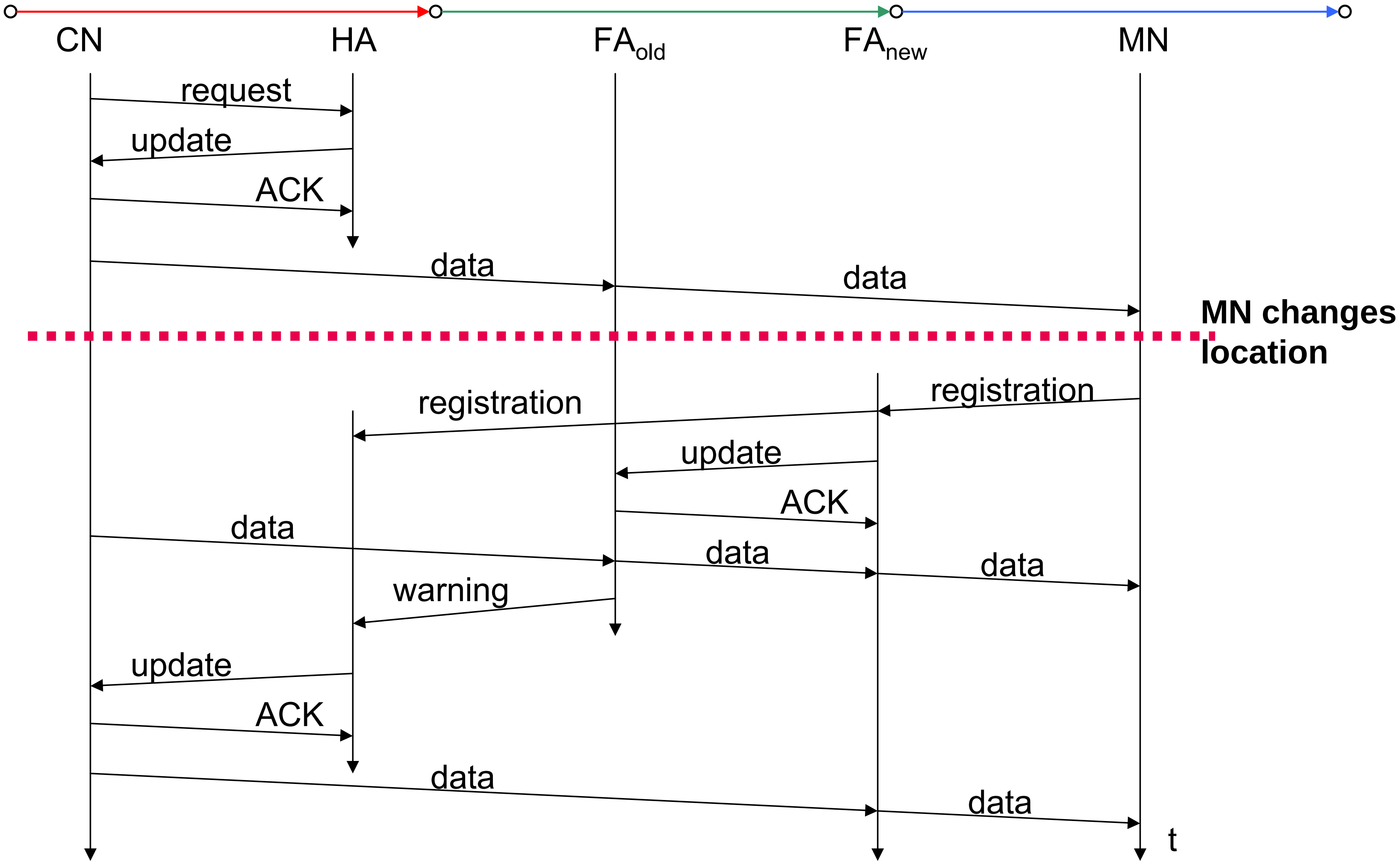
- Router accept often only “topologically correct“ addresses (firewall!)
  - a packet from the MN encapsulated by the FA is now topologically correct
  - furthermore multicast and TTL problems solved (TTL in the home network correct, but MN is too far away from the receiver)
- Reverse tunneling does not solve
  - problems with *firewalls*, the reverse tunnel can be abused to circumvent security mechanisms (tunnel hijacking)
  - optimization of data paths, i.e. packets will be forwarded through the tunnel via the HA to a sender (double triangular routing)
- Reverse tunneling is backwards compatible
  - the extensions can be implemented easily and cooperate with current implementations without these extensions

Optimization of packet forwarding



- **Triangular Routing**
  - sender sends all packets via HA to MN
  - higher latency and network load
- **"Solutions"**
  - sender learns the current location of MN
  - direct tunneling to this location
  - HA informs a sender about the location of MN
  - big security problems
- **Change of FA**
  - packets on-the-fly during the change can be lost
  - new FA informs old FA to avoid packet loss, old FA now forwards remaining packets to new FA
  - this information also enables the old FA to release resources for the MN

Change of foreign agent



Location services

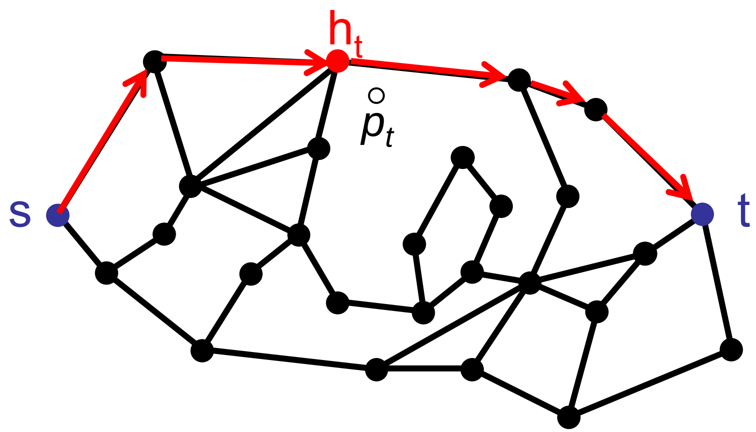
- **Service that maps node names to (geographic) coordinates**
  - Should be distributed (no require for specialized hardware)
  - Should be efficient
- **Lookup of the position (or COA) of a mobile node**
  - Mobile IP: Ask home agent
  - Home agent is determined through IP (unique ID) of MN
  - Possibly long detours even though sender and receiver are close
  - OK for Internet applications, where latency is (normally) low
- **Other application: Routing in a MANET**
  - MANET: mobile ad hoc network
  - No dedicated routing hardware
  - Limited memory on each node: cannot store huge routing tables
  - Nodes are mostly battery powered and have limited energy
  - Nodes route messages, e.g. using georouting

Home based georouting in a MANET

- **How can the sender learn the current position of another node?**
  - Flooding the entire network is undesirable (traffic and energy overhead)
- **Home based approach**
  - Similar to Mobile IP, each node has a *home* node, where it stores and regularly updates its current position
  - The home is determined by the unique ID of the node t. One possibility is to hash the ID to a position p_t and use the node closest to p_t as home.
  - Thus, given the ID of a node, every node can determine the position of the corresponding home.

Home based routing

1. Route packet to h_t, the home of the destination t
2. Read the current position of t
3. Route to t

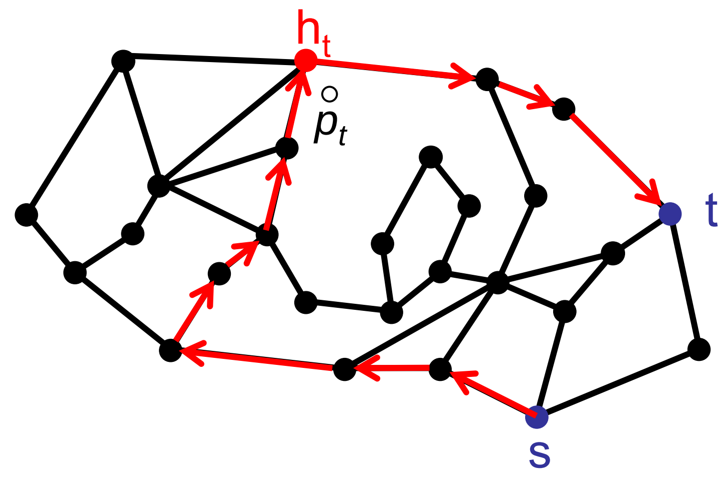


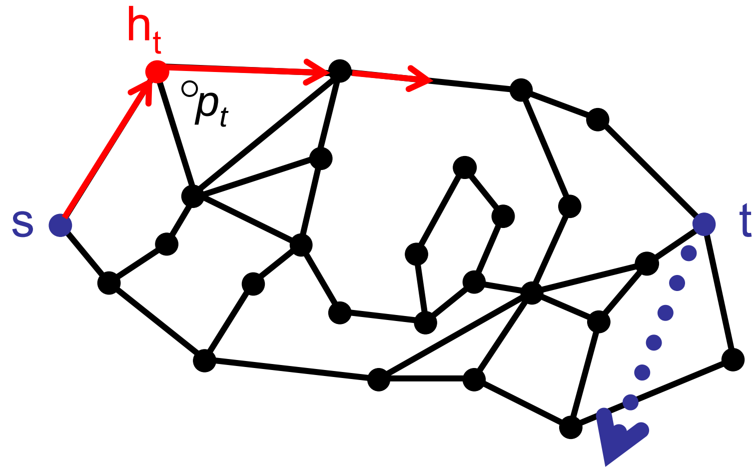
Home based location service – how good is it?

- Visiting the home of a node might be wasteful if the sender and receiver happen to be close, but the home far away
- The routing stretch is defined as

$$\text{stretch} := \frac{\text{length of route}}{\text{length of optimal route}}$$

  We want routing algorithms with low stretch.
- Simultaneous message routing and node movement might cause problems
- Can we do better?





Classification of location services

- **Proactive**
  - Mobile node divulges its position to all nodes whenever it moves
  - E.g. through flooding
- **Reactive**
  - Sender searches mobile host only when it wants to send a message
  - E.g. through flooding
- **Hybrid**
  - Both, proactive and reactive.
  - Some nodes store information about where a node is located
  - Arbitrarily complicated storage structures
  - Support for simultaneous routing and node mobility

Location services: Lookup & Publish

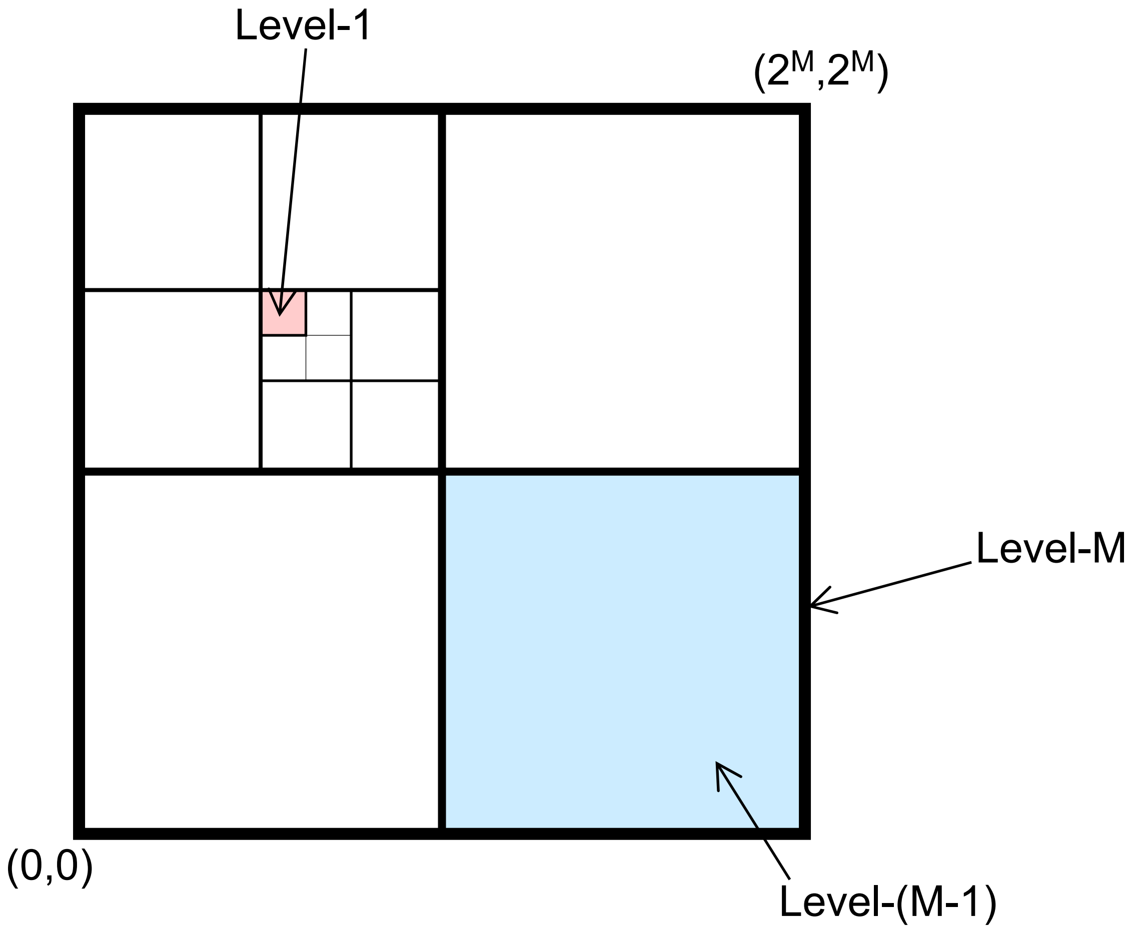
- **Any node *A* can invoke to basic operations:**
  - Lookup(*A, B*): *A* asks for the position of B
  - Publish(*A*, *x*, *y*): *A* announces its move from position *x* to *y*
- **Open questions**
  - How often does a node publish its current position?
  - Where is the position information stored?
  - How does the lookup operation find the desired information?

The Grid Location Service (GLS), Li et. al (2000)

- Cannot get reasonable stretch with one single home. Therefore, use several homes (**location servers)** where the node publishes its position.

- The location servers are selected based on a grid structure:
  - The area in which the nodes are located is divided into squares
  - All nodes agree on the lower left corner (0,0) and upper right corner (2^M, 2^M), which forms the square called **level-M**
  - Recursively, each level-N square is split into 4 level-(N-1) squares
  - The recursion stops for level-1

The Grid



Addressing of nodes

- **Unique IDs** are generated for each node (e.g. by using a hash-function)
- ID space (all possible hash values) is **circular**
- Every node can find a **least greater** node w.r.t. the ID space (the **closest node**)

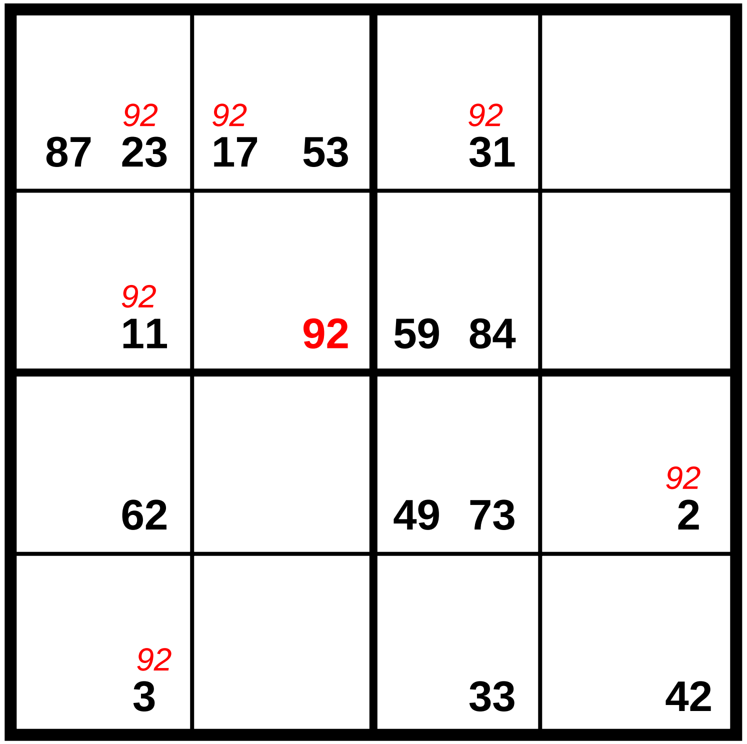
- Example:

  Let the ID space range from 1 to 99 and consider the IDs {3, 43, 80, 92}. Then, the least greater node with respect to the given ID space is

  3 → 43; 43 → 80; 80 → 92; 90 → 3

Selecting location servers

- Each node *A* recruits location servers using the underlying grid:
  - In each of the 3 level-1 squares that, along with *A*, make up a level-2 square, *A* chooses the node closest to its own ID as location server.
  - The same selection process is repeated on higher level squares.



Example for node 92, which selects the nodes {23, 17, 11} on the level-1 and {2, 3, 31} on level-2.

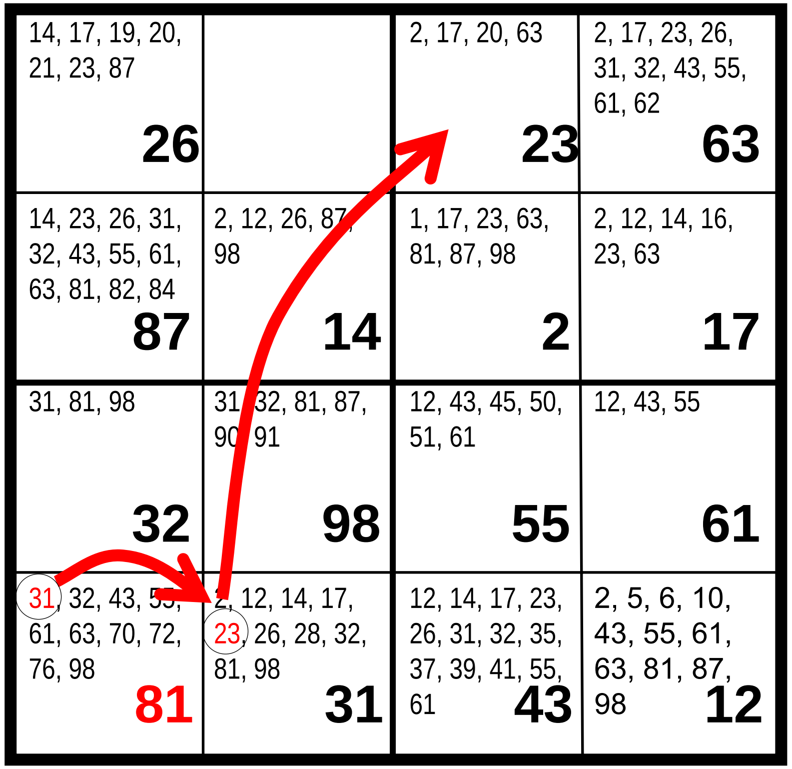
Complete example

	70, 72, 76, 81, 82, 84, 87 **90**	1, 5, 6, 10, 12, 14, 37, 62, 70, 90, 91 **16**				19, 35, 37, 45, 50, 51, 82 **39**	
1, 5, 16, 37, 62, 63, 90, 91 **70**			16, 17, 19, 21, 23, 26, 28, 31, 32, 35 **37**	19, 35, 39, 45, 51, 82 **50**		39, 41, 43 **45**	
1, 62, 70, 90 **91**	1, 5, 16, 37, 39, 41, 43, 45, 50, 51, 55, 61, 91 **62**	1, 2, 16, 37, 62, 70, 90, 91 **5**			35, 39, 45, 50 **51**		19, 35, 39, 45, 50, 51, 55, 61, 62, 63, 70, 72, 76, 81 **82**
	62, 91, 98 **1**				19, 20, 21, 23, 26, 28, 31, 32, 51, 82 **35**	1, 2, 5, 6, 10, 12, 14, 16, 17, 82, 84, 87, 90, 91, 98 **19**	
14, 17, 19, 20, 21, 23, 87 **26**		2, 17, 20, 63 **23**	2, 17, 23, 26, 31, 32, 43, 55, 61, 62 **63**	28, 31, 32, 35, 37, 39 **41**		10, 20, 21, 28, 41, 43, 45, 50, 51, 55, 61, 62, 63, 70 **72**	
14, 23, 26, 31, 32, 43, 55, 61, 63, 81, 82, 84 **87**	2, 12, 26, 87, 98 **14**	1, 17, 23, 63, 81, 87, 98 **2**	2, 12, 14, 16, 23, 63 **17**		6, 10, 20, 21, 23, 26, 41, 72, 76, 84 **28**	6, 72, 76, 84 **10**	
31, 81, 98 **32**	31, 32, 81, 87, 90, 91 **98**	12, 43, 45, 50, 51, 61 **55**	12, 43, 55 **61**	1, 2, 5, 21, 76, 84, 87, 90, 91, 98 **6**	6, 10, 20 , 76 **21**		6, 10, 12, 14, 16, 17, 19, 84 **20**
31, 32, 43, 55, 61, 63, 70, 72, 76, 98 **81**	2, 12, 14, 17, 23, 26, 28, 32, 81, 98 **31**	12, 14, 17, 23, 26, 31, 32, 35, 37, 39, 41, 55, 61 **43**	2, 5, 6, 10, 43, 55, 61, 63, 81, 87, 98 **12**		6, 21, 28, 41, 72 **76**	20, 21, 28, 41, 72, 76, 81, 82 **84**	

Querying location of other nodes

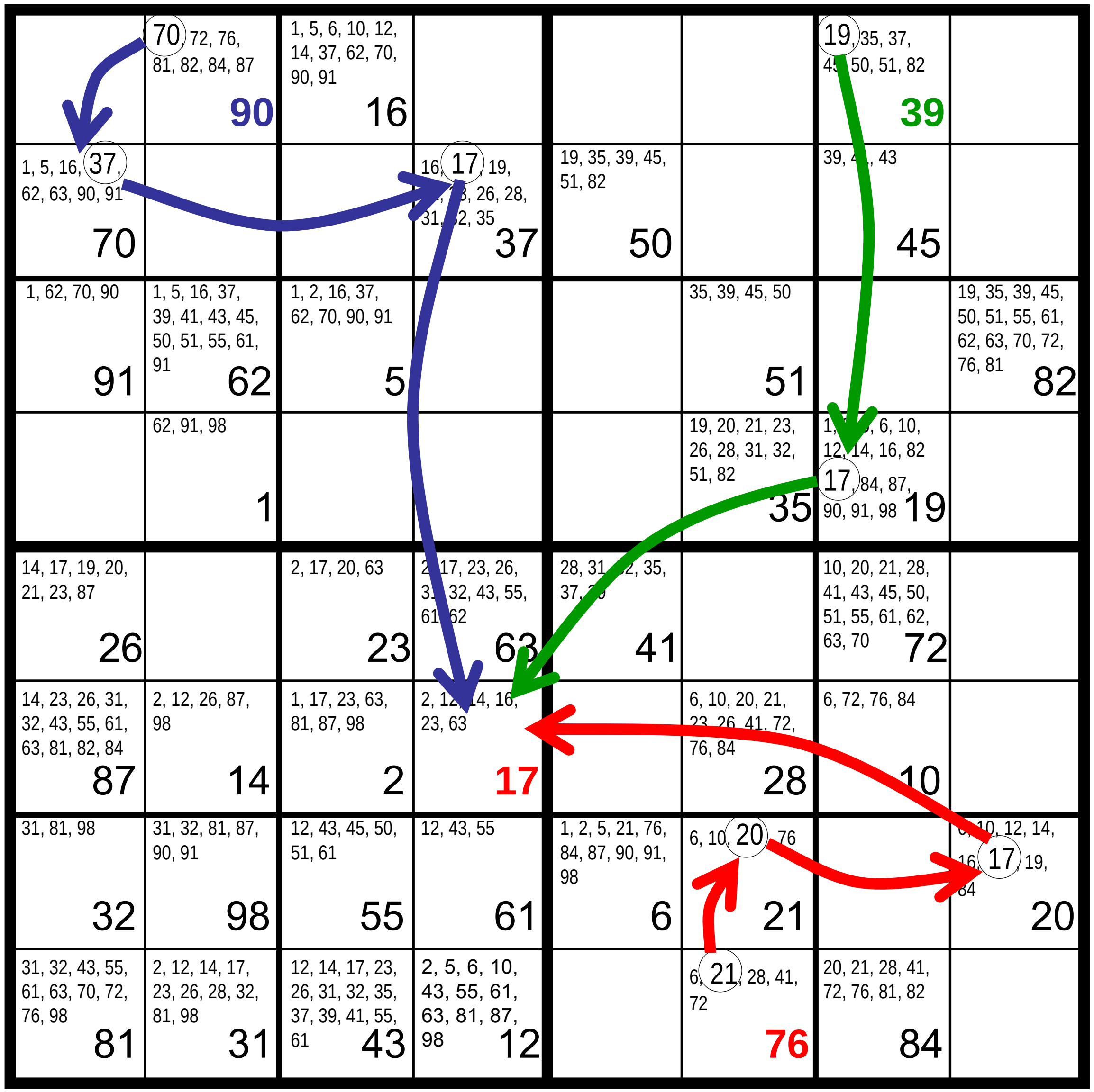
- **Lookup(*A*, *B*): Find a location server of node B**
  1. Node *A* sends the request (with georouting) to the node with ID closest to B for which *A* has location information
  2. Each node on the way forwards the request in the same way
  3. Eventually, the query reaches a location server of *B*, which forwards it to *B*.

Example: Send packet from 81 to 23



Lookup Example

Lookup for 17 from 76, 39 and 90



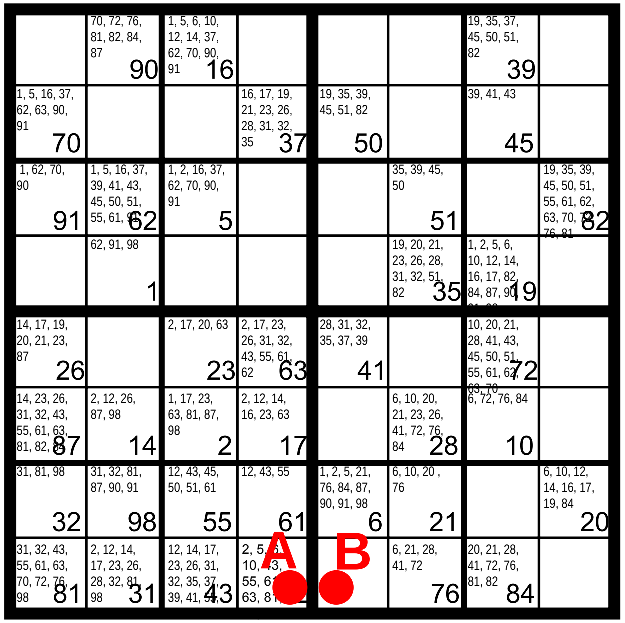
Analysis of GLS

- **Theorem 1:** A query needs no more than *k* location query steps to reach a location server of the destination when the sender and receiver are colocated in a level-*k* square.

- **Theorem 2:** The query never leaves the level-*k* square in which the sender and destination are colocated.

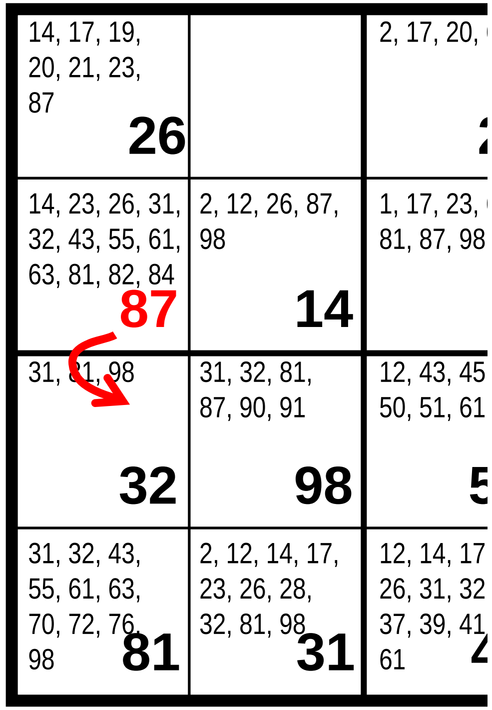
GLS has no worst case guarantees

- The lookup cost between two nodes might be arbitrarily high even though the nodes are very close
- The publish cost might be arbitrarily high even though a node only moved a very short distance
- In sparse networks, routing to the location server may have worst case cost, while routing directly can be more efficient



GLS and mobility

- Node crosses boundary line: what happens to the node's role as location server?
  - Must redistribute all information in the old level
  - Gather new information in the new level
  - Publish cost is arbitrarily high compared to the moved distance



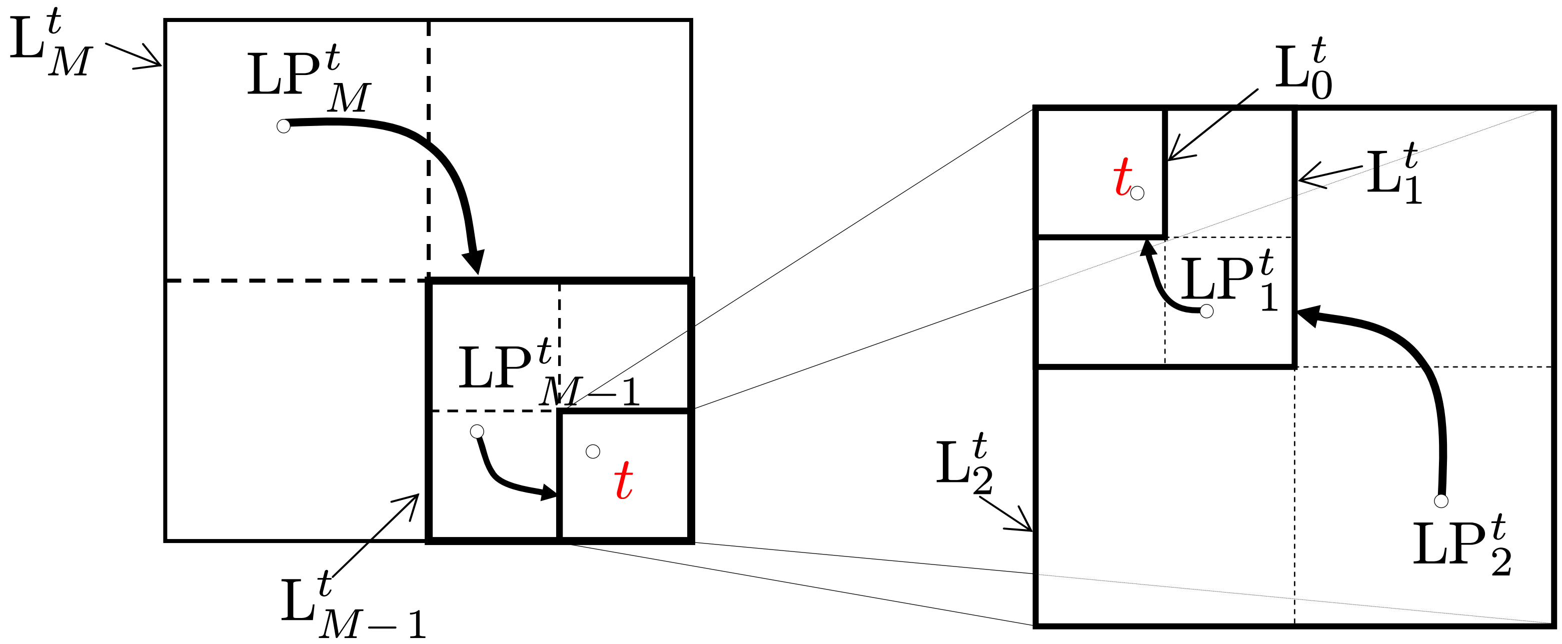
- A lookup happening in parallel with node movement might fail. Thus, GLS does not guarantee delivery for real concurrent systems, where nodes might move independently at any time.

Improving GLS

- **Goals for MLS**
  - Publish cost only depends on moved distance
  - Lookup cost only depends on the distance between the sender and receiver
  - Nodes might move arbitrarily at any time, even while other nodes issue lookup requests
  - Determine the maximum allowed node speed under which MLS still guarantees delivery

Location pointers (aka location servers)

- Difference to GLS:
  - Only *one* location pointer (LP) per level (L) (GLS: 3 location servers)
  - The location pointer only knows in which sub-level the node is located (GLS: the location server knows the exact position)



Location pointer & Notation

- Notation:
  - LP_k^t Location pointer for node *t* on level-*k*
  - L_k^t Level-*k* that contains node *t*

- The location pointers are placed depending on their ID, as in the home-based lookup system.

- The position of LP_k^t is obtained by hashing the ID of node *t* to a position in L_k^t. The location pointer is stored on the nearest nodes.

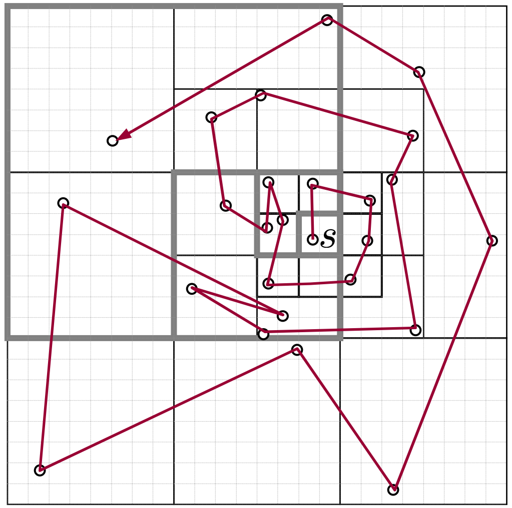
Routing in MLS

- Routing from a node *s* to a node *t* consists of two phases:
  1. Find a location pointer LP_k^t
  2. Once a first location pointer is found on level-*k*, we know in which of the 4 sub-squares *t* is located and thus in which L_{k-1} *t* has published another location pointer LP_{k-1}^t .
     Recursively, the message is routed towards location pointers on lower levels until it reaches the lowest level, from where it can be routed directly to *t*.

Routing in MLS (2)

- When a node *s* wants to find a location pointer of a node *t*, it first searches in its immediate neighborhood and then extends the search area with exponential growing coverage.
  - First, try to find a location pointer LP_0^t in L_0^s or one of its 8 neighboring levels.
  - Repeat this search on the next higher level until a LP_k^t is found

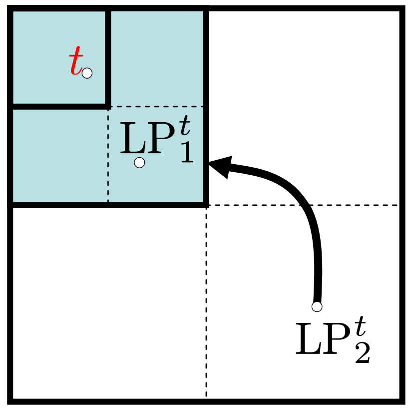
- The lookup path draws a spiral-like shape with exponentially increasing radius until it finds a location pointer of *t*.
- Once a location pointer is found, the lookup request knows in which sub-square it can find the next location pointer of *t*.

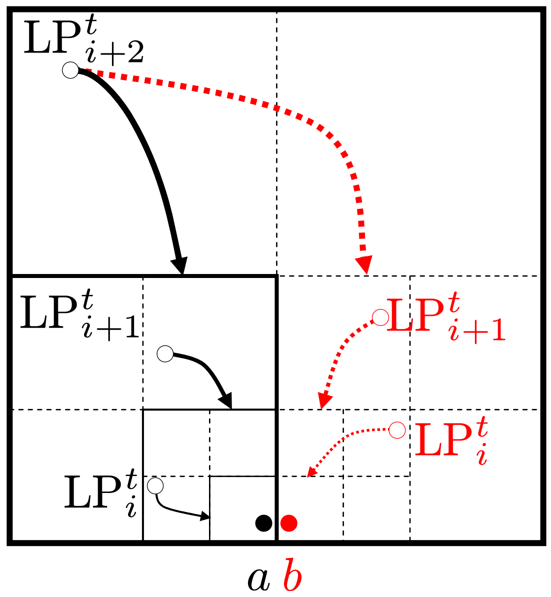


Support for mobility in MLS

- A location pointer only needs to be updated when the node leaves the corresponding sub-square.
  - LP_2^t is OK as long as t remains in the shaded area.
  - Most of the time, only the closest few location pointers need to be updated due to mobility.

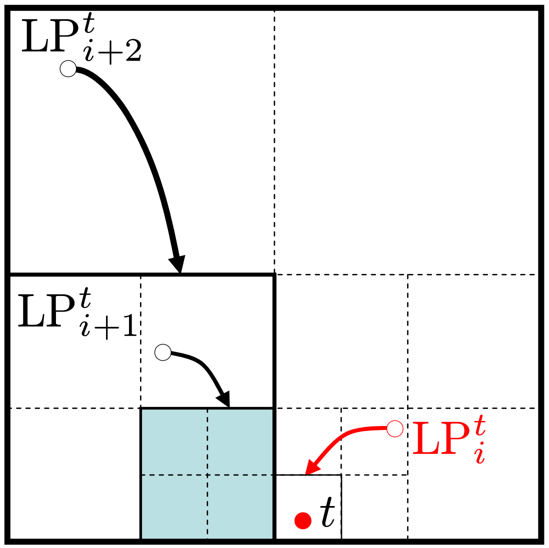
- Not enough: If a node moves across a level boundary, many pointers need to be updated. E.g. a node oscillates between the two points a and b.





Lazy publishing

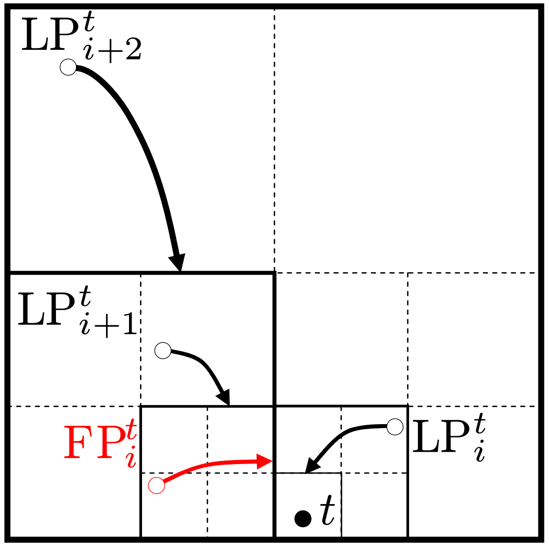
- Idea: Don't update a level pointer LP_k^t as long as *t* is still somewhat close to the level L_k where LP_k^t points.



- Breaks the lookup: LP_{i+1}^t points to a level that does not contain LP_i^t

Lazy publishing with forwarding pointers

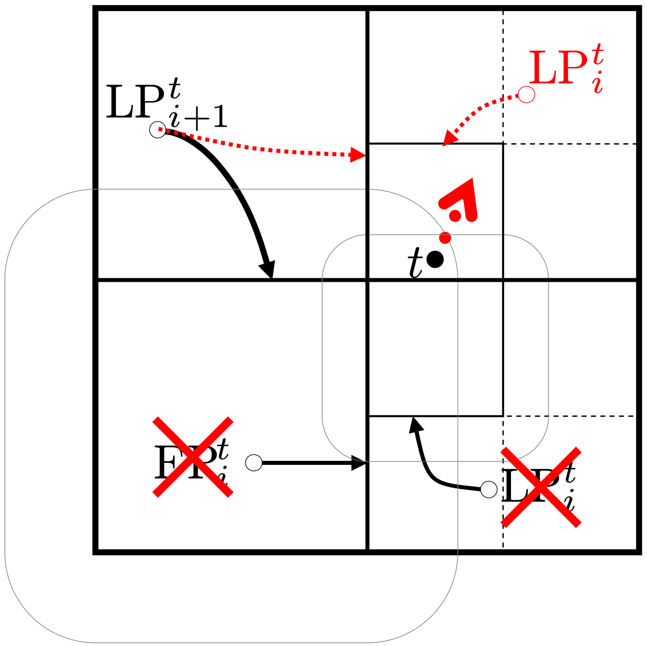
- No problem, add a **forwarding pointer** that indicates in which neighboring level the location pointer can be found.



Concurrency in MLS

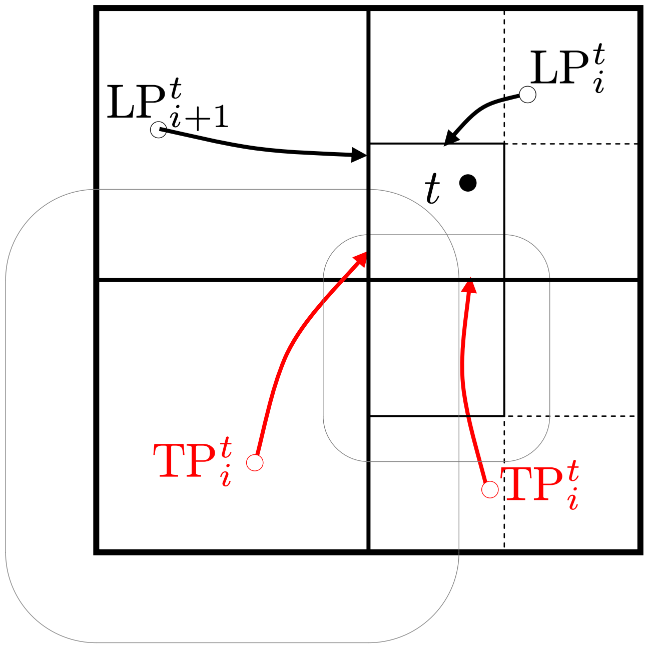
- Allowing for concurrent lookup requests and node mobility is somewhat tricky, especially the deletion of pointers.
- Note that a lookup request needs some time to travel between location pointers. The same holds for requests to create or delete location (or forwarding) pointers.

- Example:
  - A lookup request follows LP^t_{i+1}, and node *t* moves as indicated
  - *t* updates its LP^t_i and LP^t_{i+1} and removes the FP^t_i and the old LP^t_i
  - The lookup request fails if it arrives after the FP^t_i has been removed



Concurrency in MLS (2)

- No problem either: Instead of removing a location pointer or forwarding pointer, replace it with a **temporary pointer** that remains there for a *short time* until we are sure that no lookup request might arrive anymore on this outdated path.
- Similar to the forwarding pointer, a temporary pointer redirects a lookup to the neighbor level where the node is located.



Properties of MLS

- Constant lookup stretch
  - The length of the chosen route is only a constant longer than the optimal route
- Publish cost is O(*d* log *d*) where moved distance is *d*
  - Even if nodes move considerably, the induced message overhead due to publish requests is moderate.
- Works in a concurrent setup
  - Lookup requests and node movement might interleave arbitrarily
- Nodes might not move faster than 1/15 of the underlying routing speed
  - We can determine the maximum node speed that MLS supports. Only if nodes move faster, there might arise situations where a lookup request fails.

MLS Conclusions

- It’s somewhat tricky to handle concurrency properly
  - Use of temporary forwarding pointers
- MLS is the first location service that determines the maximum speed at which nodes might move
  - Without the speed limitation, no delivery guarantees can be made!
- Drawbacks
  - MLS utilizes an underlying routing algorithm that can deliver messages with constant stretch given the position of the destination
  - MLS requires a relatively dense node population