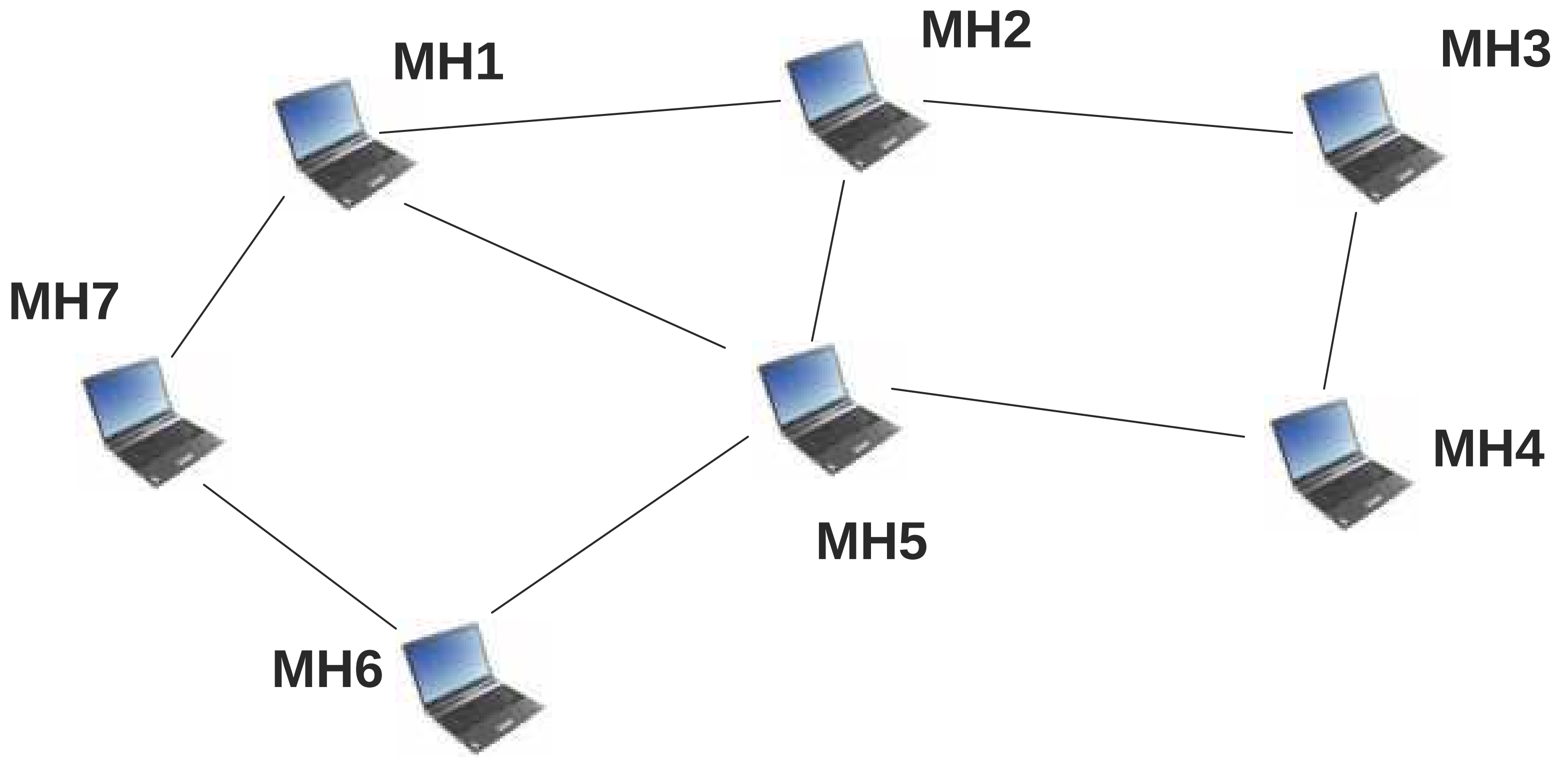
Mobile and Ubiquitous Computing
Routing Protocols

Niki Trigoni
www.dcs.bbk.ac.uk/~niki
niki@dcs.bbk.ac.uk

Overview

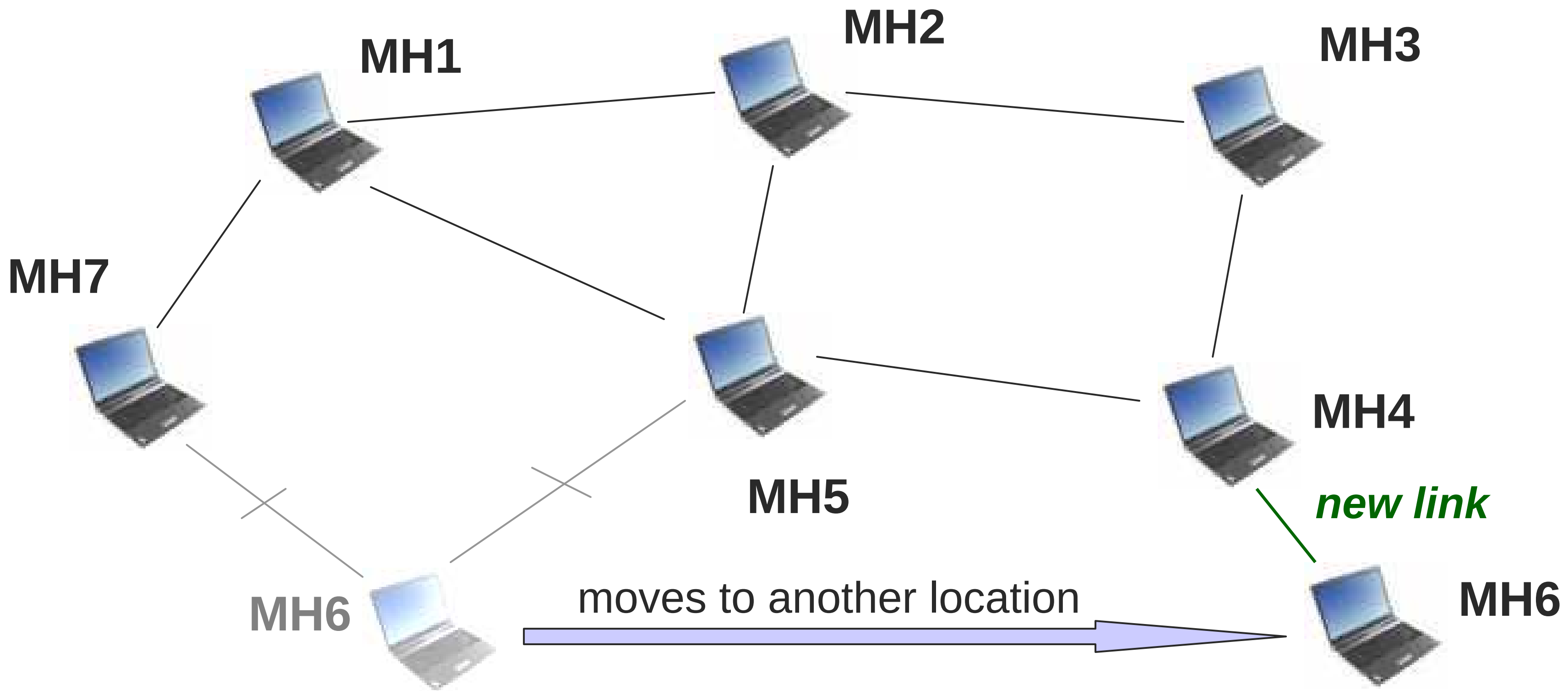
- Intro to routing in ad-hoc networks
- Routing methods
  - Link-State
  - Distance-Vector
- Distance-vector routing protocols
  - DSDV (proactive)
  - AODV (reactive)

Routing in ad-hoc networks



- Adhoc network of mobile hosts represented as a graph G(N,E(t))
- Two nodes are connected with an edge if they are within communication range

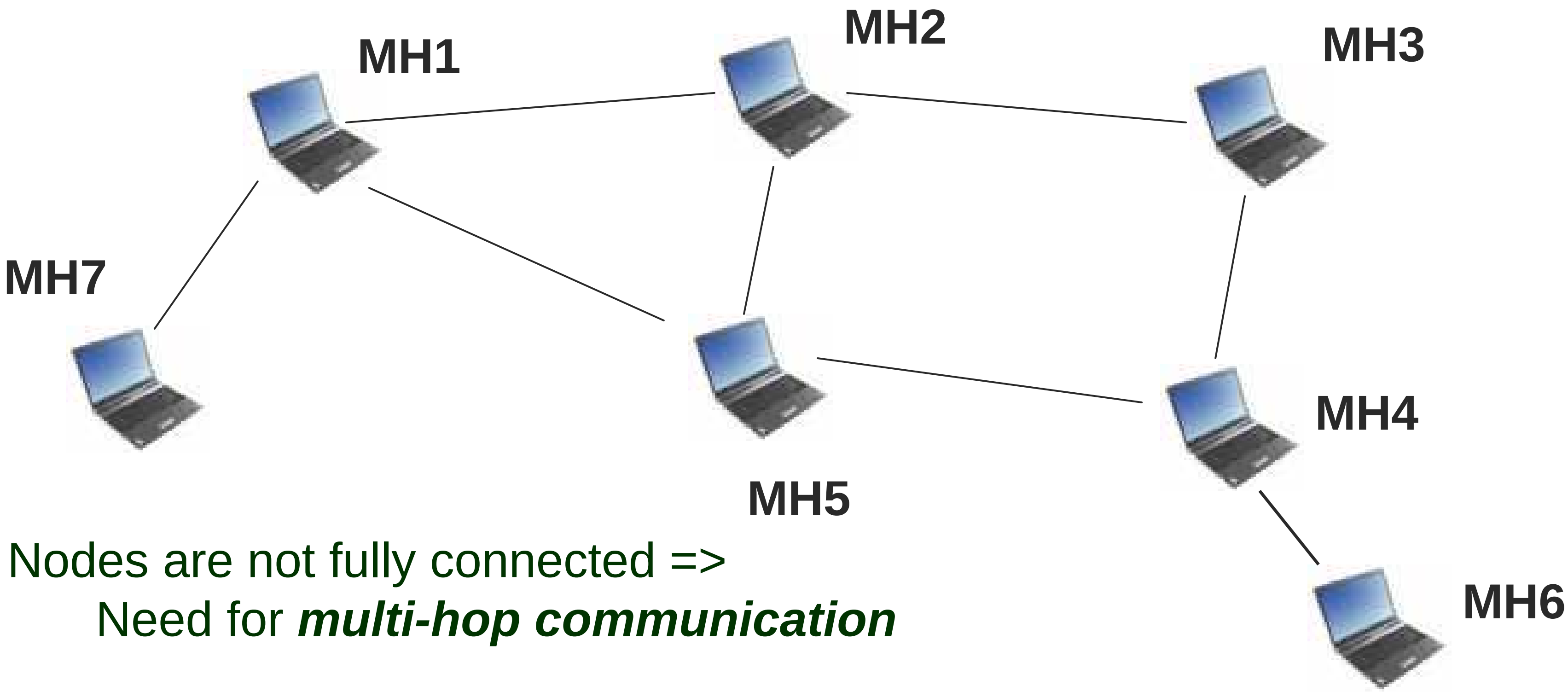
Routing in ad-hoc networks



- Mobile hosts can move
- Mobile hosts can be dynamically added or removed from the network

=> The network connectivity changes dynamically

Routing in ad-hoc networks

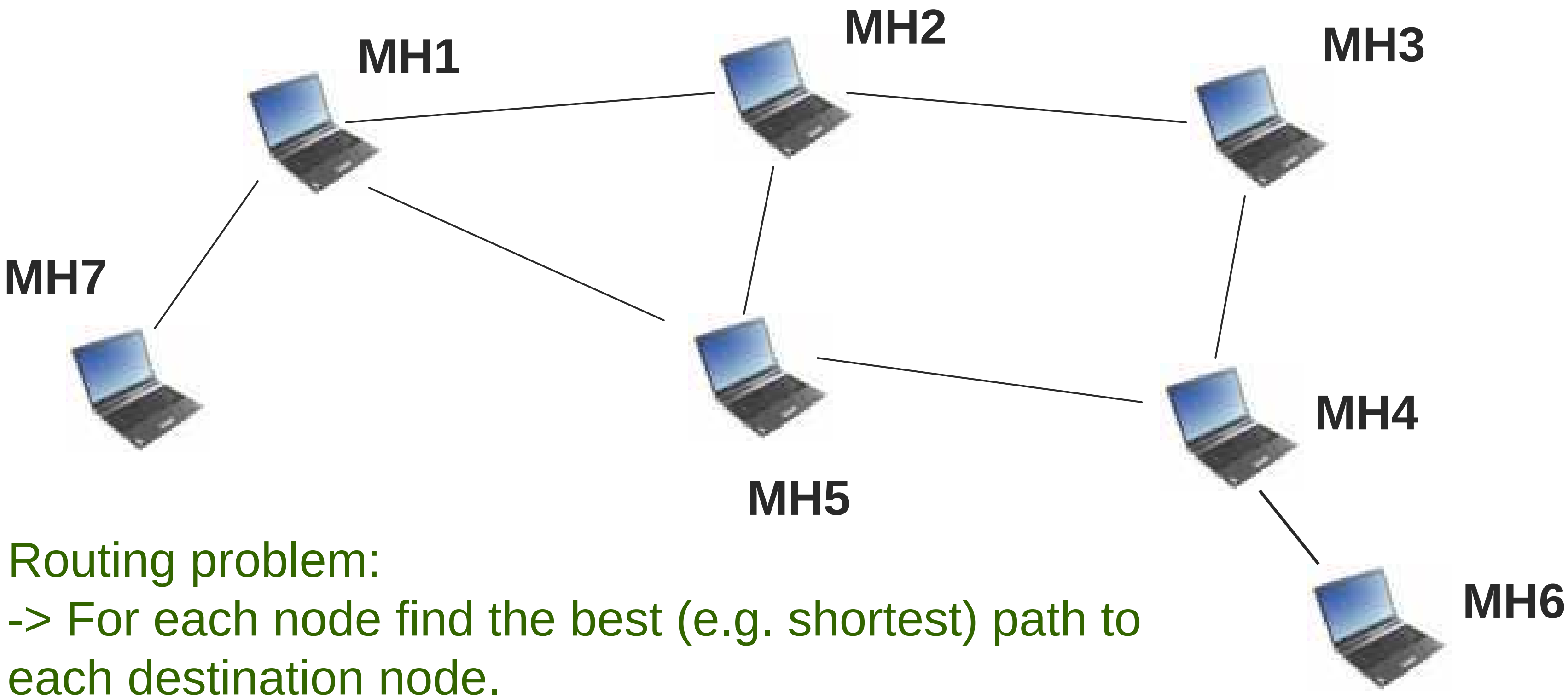


Nodes are not fully connected =>
Need for ***multi-hop communication***

Say MH7 wants to send a message to MH3. Several options:

- MH7 -> MH1 -> MH2 -> MH3
- MH7 -> MH1 -> MH5 -> MH2 -> MH3
- MH7 -> MH1 -> MH5 -> MH4 -> MH3 etc.

Routing in ad-hoc networks



Routing problem:
-> For each node find the best (e.g. shortest) path to each destination node.

Distributed version of the routing problem:
-> For each node find the next hop in the best (e.g. shortest) path to each destination node.

Routing in ad-hoc networks

- Each node first identifies the preferred neighbor (next hop) in the optimal path to each destination.
- A data packet is forwarded hop-by-hop from the source to the destination along the optimal path:
  - The data packet contains the destination node in its header.
  - When a node receives a data packet, it forwards it to the preferred neighbor for its destination.

Link-state vs. distance-vector

Link-state approach:

- Each node has a complete view of the network topology
- Each node propagates the costs of its outgoing links to all other nodes

Distance-vector approach (Distributed Bellman-Ford):

- Every node **i** maintains for each destination x a set of distances $d_{ij}(x)$ for each neighbor node **j**: $d_{ij}(x)$ is the cost (e.g. number of hops) of sending a data packet to **x** through neighbor **j**
- Node **i** selects to forward a data packet through neighbor **k** such that: $d_{ik}(x) = \min_j \{d_{ij}(x)\}$
- Each node periodically broadcasts to its neighbors its current estimate of the shortest distance to every destination node.

Link-state vs. distance-vector

Problems of the link-state approach:

- Requires large storage space and heavy computation
- Inconsistent views of network topologies

  => short-lived routing loops

Problems of the distance-vector approach:

- More efficient than link-state in terms of computation and storage requirements
- Stale routing information causes routing loops

Nodes choose their next hops in a distributed manner → short-lived and long-lived routing loops

DSDV: Destination-Sequenced Distance-Vector protocol

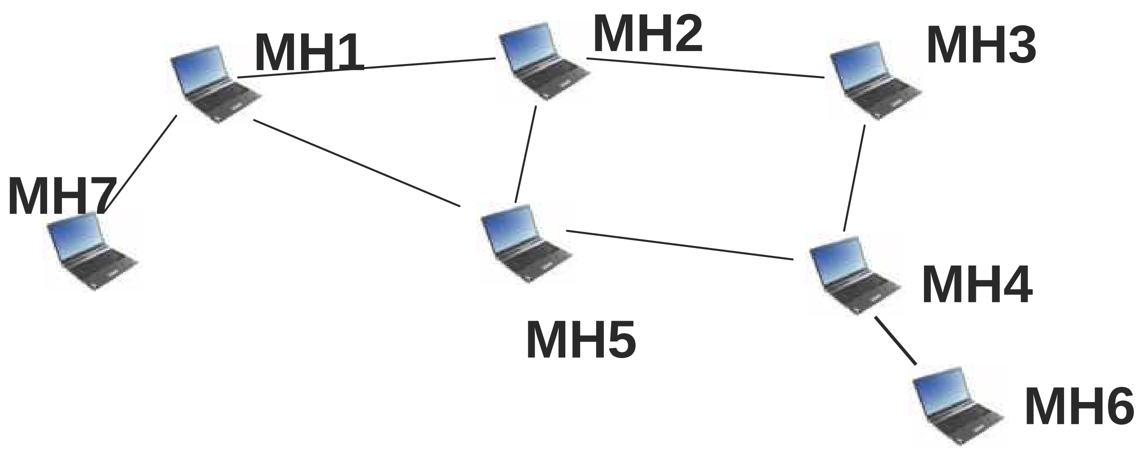
- Each node maintains locally a routing table
- Each entry of the routing table includes routing information for a destination node:
  - the next hop in the optimal path to the destination
  - the cost of the optimal path to the destination
  - the freshness (sequence no) of the path to the destination
- The node advertises the local routing table to its neighbors
  - Periodically
  - When topology changes are detected
- On receiving routing information from a neighbor, a node uses it to update its own local routing table

DSDV: Destination-Sequenced Distance-Vector protocol

Sequence number is generated at the destination

A few entries in MH1's routing table

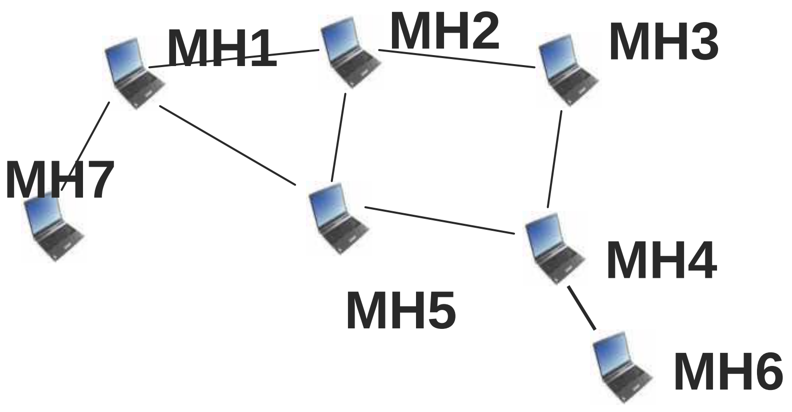
Destination	Next Hop	Metric	Sequence Number	Install	Stable Data
MH2	MH2	1	S212_MH2	…	…
MH3	MH2	2	S302_MH3		
MH4	MH5	2	S100_MH4	…	…



DSDV: Destination-Sequenced Distance-Vector protocol

MH1 Routing Table

Destin	Next Hop	Metric	Sequence Number
MH6	MH2	4	S200_MH6
…	…	…	…
…	…	…	…



What if MH1 receives new routing information **(Dest=MH6, Metric=2, SeqNo=S200_MH6)** from MH5 ?

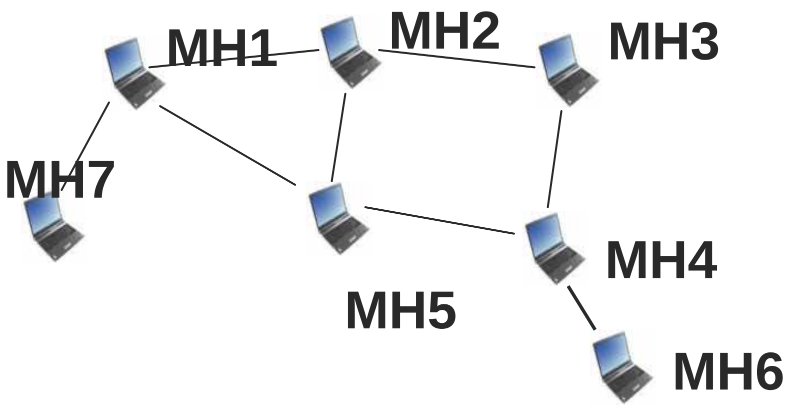
MH1 Routing Table (updated)

Destin	Next Hop	Metric	Sequence Number
MH6	**MH5**	**3**	S200_MH6
…	…	…	…
…	…	…	…

DSDV: Destination-Sequenced Distance-Vector protocol

MH1 Routing Table

Destin	Next Hop	Metric	Sequence Number
MH6	MH5	3	S200_MH6
…	…	…	…
…	…	…	…

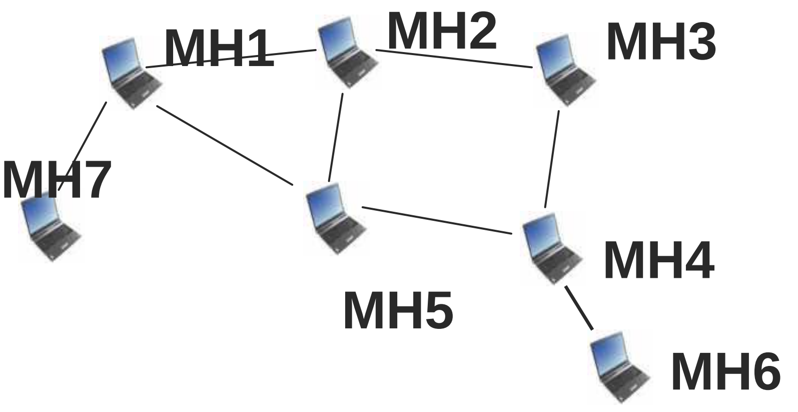


Any routing information that MH1 receives regarding Dest=MH6 that has sequence number smaller than 200 (S200_MH6) is considered stale, and it is ignored by MH1.

DSDV: Destination-Sequenced Distance-Vector protocol

MH1 Routing Table

Destin	Next Hop	Metric	Sequence Number
MH6	MH5	3	S200_MH6
…	…	…	…
…	…	…	…



MH1 Routing Table (updated)

Destin	Next Hop	Metric	Sequence Number
MH6	MH5	3	**S201_MH6**
…	…	…	…
…	…	…	…

What if MH1 receives new routing information

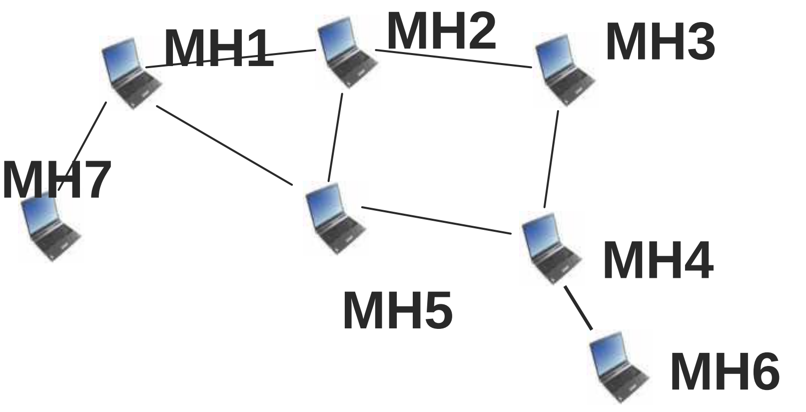
(Dest=MH6, Metric=2, SeqNo=S201_MH6)

from MH5 ?

DSDV: Destination-Sequenced Distance-Vector protocol

MH1 Routing Table

Destin	Next Hop	Metric	Sequence Number
MH6	MH5	3	S200_MH6
…	…	…	…
…	…	…	…



MH1 Routing Table (updated)
New routing entry is broadcast

Destin	Next Hop	Metric	Sequence Number
MH6	MH5	∞	S201_MH6
…	…	…	…
…	…	…	…

What if the link between MH1 and MH5 breaks?

DSDV: Destination-Sequenced Distance-Vector protocol

- Compare new routing information with the information available in the local routing table
- Prefer routes with more recent sequence numbers
- Discard routes with older sequence numbers
- Prefer routes with sequence number equal to an existing entry if it has a better metric value
- Newly recorded routes are scheduled for immediate broadcasting
- Updated routes only with a new sequence number are scheduled for advertisement at a later time

DSDV: Destination-Sequenced Distance-Vector protocol

- Two modes of propagating routing information:
  - **Full dump:** All available routing information is broadcast
  - **Incremental dump:** Only information changed since the last full dump is broadcast

- When mobile nodes do not move a lot, full dumps are sent infrequently.
- When the network topology changes fast, full dumps are scheduled more frequently.

AODV: Ad Hoc On-Demand Distance-Vector protocol

- Compared to DSDV, AODV tries to reduce the number of broadcasts resulting from changes in network topology
  - In DSDV, local movements have global effects
  - In AODV, non-local effects are limited to nodes trying to reach a distant node through a broken link

AODV: Ad Hoc On-Demand Distance-Vector protocol

- AODV
  - does not maintain routes from every node to every other node in the network.
  - discovers routes on-demand (reactively, not proactively)
  - provides unicast, multicast and broadcast communication ability
  - uses two route tables
    - for unicast routes and
    - for multicast routes
- We will consider only unicast route discovery.

AODV: Ad Hoc On-Demand Distance-Vector protocol

Unicast routing

A node wishes to send a packet to a destination node D. It first checks whether it has a valid route to D.

- If yes, it sends the packet to the next hop towards the destination.
- If not, it initiates a ***route discovery process***.

AODV: Ad Hoc On-Demand Distance-Vector protocol

Unicast routing: Route Discovery Process

- The node creates a RREQ (RouteRequest) packet
  - sourceIPAddress
  - sourceBroadcastId
  - destIPAddress
  - lastKnownSequenceNo
  - hopCount
- The node broadcasts the RREQ
- The node sets a timer to wait for a reply

AODV: Ad Hoc On-Demand Distance-Vector protocol

Unicast routing: Route Discovery Process

- When a node receives a RREQ, it ignores it if it has seen another routing packet with the same <sourceIPAddress, sourceBroadcastId> pair.
- Otherwise, the node sets up a reverse routing entry in its routing table:
  - sourceIPAddress
  - sourceBroadcastIP
  - hopsToSource
  - prevHopToSource
- Route entries that exceed their lifetime are deleted.

AODV: Ad Hoc On-Demand Distance-Vector protocol

Unicast routing: Route Discovery Process

- A node responds to an RREQ if it has
  - an unexpired entry for the destination in its route table
  - with sequence no >= RREQ's lastKnownSequenceNo

  By unicasting a RREP back to the source.

- If a node cannot respond to an RREQ, it increments the RREQ's hop count and then broadcasts the packet to its neighbors.

AODV: Ad Hoc On-Demand Distance-Vector protocol

Unicast routing: Route Replies (RREPs)

- If an intermediate node is responding to a RREQ, it populates the RREP as follows:
  - It places its record of the destination's sequence number in the packet
  - sets the hop count equal to its distance from the destination
  - Initializes the RREP's lifetime

AODV: Ad Hoc On-Demand Distance-Vector protocol

Unicast routing: Forward Path Setup

On receiving an RREP, a node:

- sets up a forward path entry to the destination
  - destinationIPAddress
  - IPOfNeighborWhoSentRREP
  - hopCountToDestination
  - routingEntryLifetime

- Each time a route is used the associated lifetime is updated in the routing table

Summary

- Two distinct approaches to routing:
  - Proactive: nodes continuously maintain routes to all destination, even if they don't use them frequently (DSDV).
  - Reactive: nodes identify and maintain routes on-demand, i.e. when they need to send packets to a certain destination (AODV).
- Both DSDV and AODV are distance-vector protocols:
  - Nodes maintain distances (costs) to destinations and keep information about the next hop in the optimal path to a destination.
- Both DSDV and AODV are designed for adhoc (wireless mobile) networks

Related Reading

- C.E. Perkins and P. Bhagwat. Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers. In ACM SIG-COMM Computer Communications Review 24(4), pages 234-244, October 1994

Paper to prepare for discussion:

- C.E. Perkins and E.M. Royer. *Ad Hoc On-Demand Distance-Vector Routing*. In Proceedings of the Second Annual IEEE Workshop on Mobile Computing Systems and Applications, February 1999, pages 90-100.

TinyOS Tutorial – Lab 1

- Platform specification
- TinyOS
- NesC
- Examples

Platform specification

- 8 MHz Processor
- 10k RAM, 48k Flash
- 250kbps 2.4GHz Radio
  - 50m range indoors / 125m range outdoors
- Integrated Humidity, Temperature, and Light sensors
- Programming and data collection via USB
- TinyOS support

TinyOS

- Event-driven OS
  - Tasks
  - Events
- Tasks **cannot** interrupt other tasks or events
- Events **can** interrupt other tasks or events
  - Concurrency issues (`atomic` statement)

NesC

- **Application**
  - A NesC application consists of one or more components, linked together
- **Component**
  - Components are of two types: modules and configurations
- **Module**
  - A module contains the application code in a C-like syntax
- **Configuration**
  - A configuration wires components together
- **Interface**
  - An interface specifies a set of available functions

Interfaces

- Components are wired through interfaces
- An interface can either be **provided** or **used** by a component

Provided interfaces

- When providing an interface
  - All the **commands** have to be implemented
  - All the **events** should be called
- Example: `Leds` interface and `LedsC` component

Leds.nc and LedsC.nc

```
interface Leds {
  command result_t redOn();
  command result_t redOff();
  command result_t redToggle();
  command result_t greenOn();
  // …
}
```

```
module LedsC {
  provides interface Leds;
}
implementation {
  command result_t redOn()
      {
    // …
  }
  command result_t redOff() {
    // …
  }
  // …
}
```

Used interfaces

- When using an interface
  - All the commands can be called
  - All the events have to be implemented

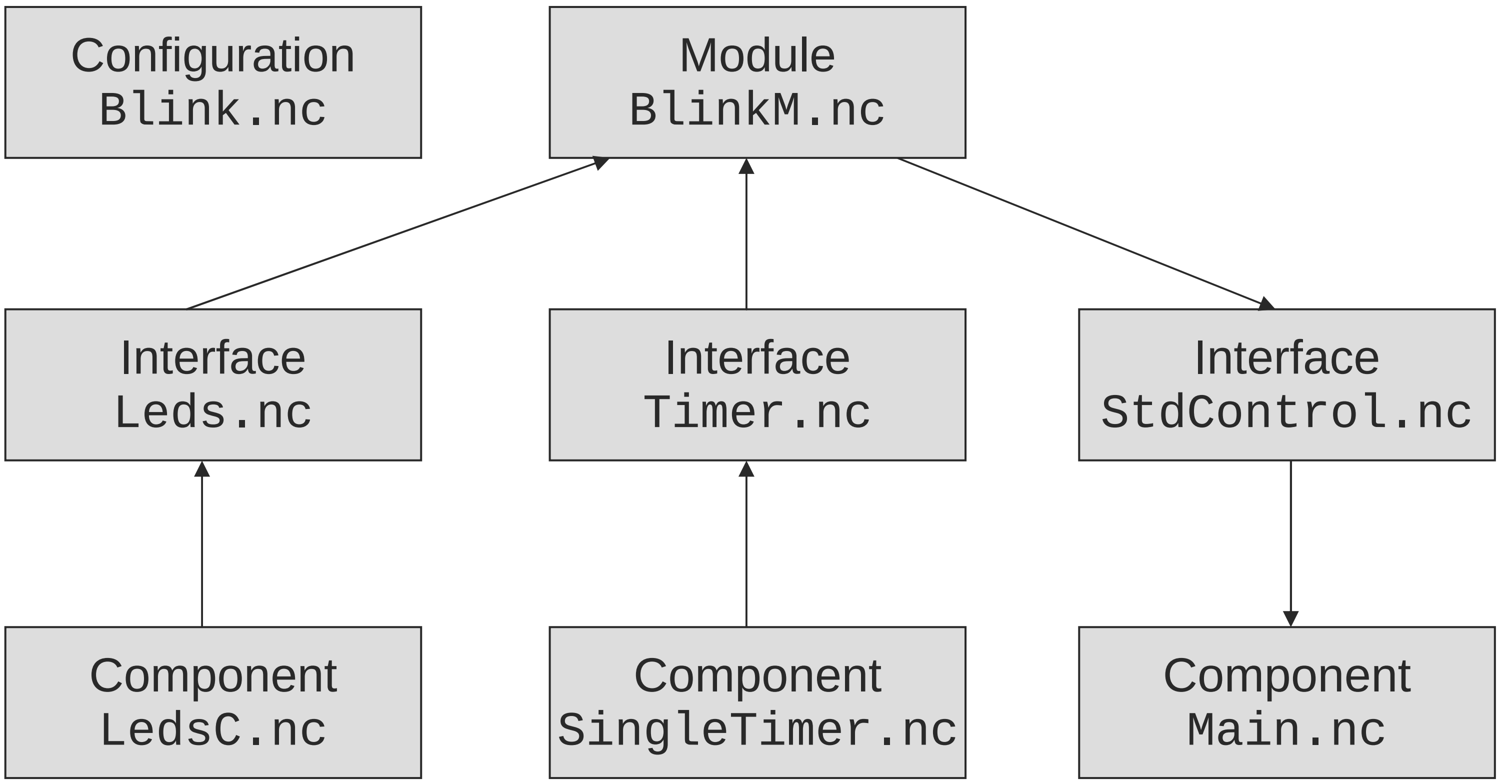
- Example: `Timer` interface and `BlinkM` component

Timer.nc and BlinkM.nc

```
interface Timer {
 command result_t
       start(char, uint32_t);
  command result_t stop();
  event result_t fired();
}
```

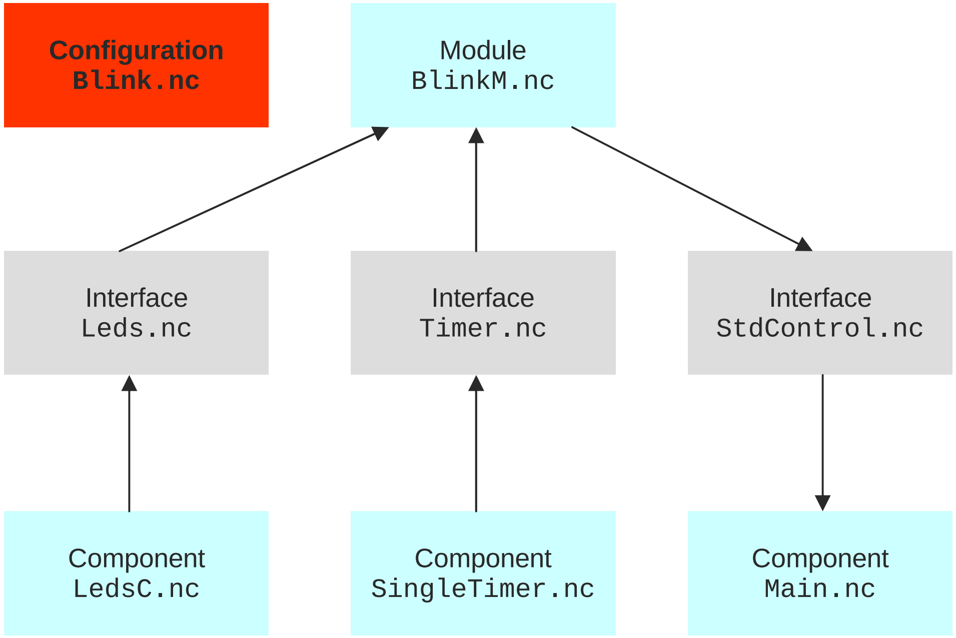
```
module BlinkM {
  use {
    interface Leds;
    interface Timer;
  }
}
implementation {
  event result_t Timer.fired()
  {
    call Leds.redToggle();
    return SUCCESS;
  }
  // …
}
```

Blink application



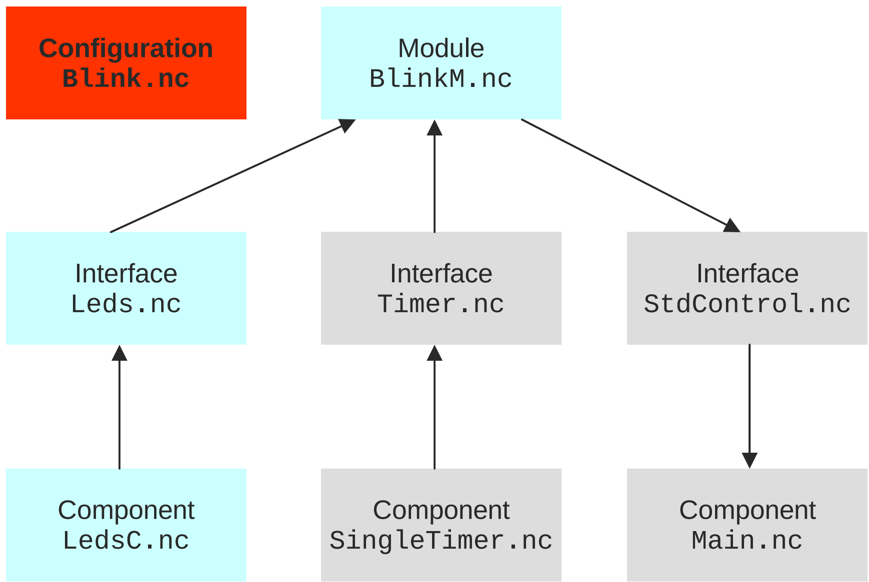
Blink.nc

```
configuration Blink {
}
implementation {
  components BlinkM, LedsC,
      SingleTimer, Main;

  BlinkM.Leds -> LedsC;
  // or BlinkM.Leds -> LedsC.Leds
  BlinkM.Timer -> SingleTimer;
  Main.StdControl -> BlinkM;
}
```



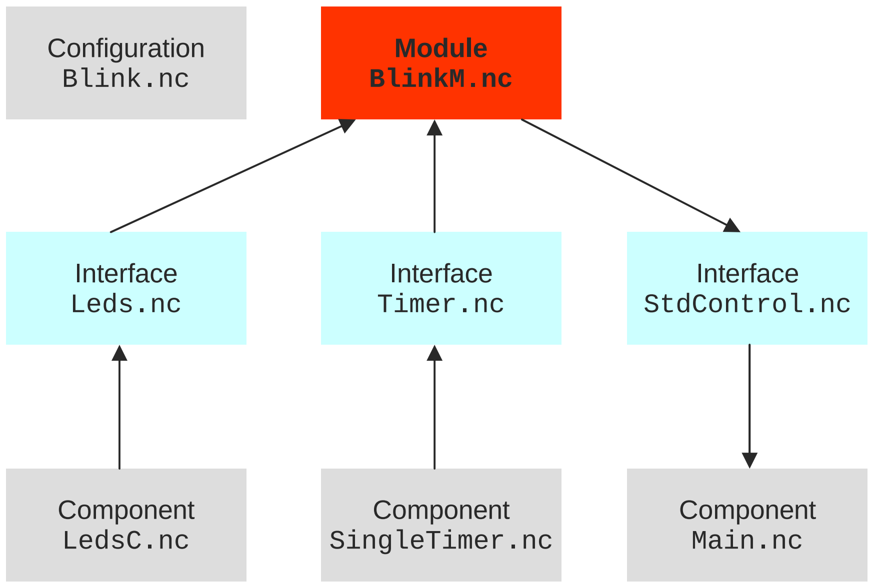
Blink.nc

```
configuration Blink {
}
implementation {
  components BlinkM, LedsC,
      SingleTimer, Main;

  BlinkM.Leds -> LedsC;
  // or BlinkM.Leds -> LedsC.Leds
  BlinkM.Timer -> SingleTimer;
  Main.StdControl -> BlinkM;
}
```



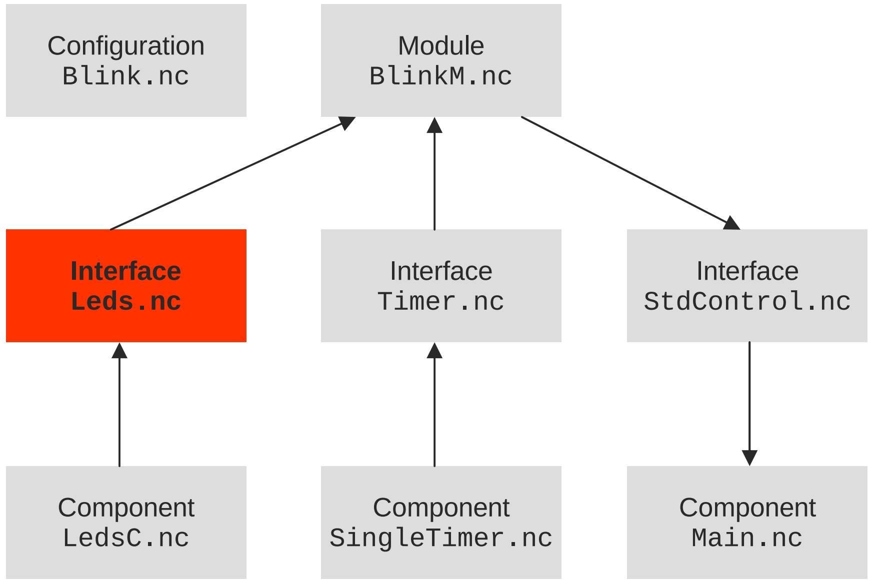
BlinkM.nc

```
module BlinkM {
    uses {
        interface Leds;
        interface Timer;
    }
    provides {
        interface StdControl;
    }
}

implementation {
  // …
}
```



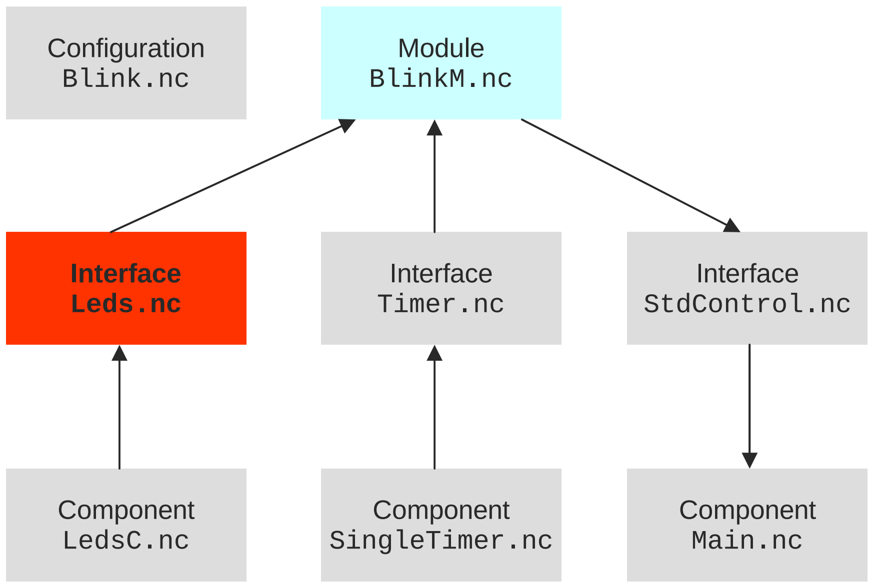
Leds.nc

```
interface Leds {
  command result_t redOn();
  command result_t redOff();
  command result_t redToggle();
  // …
}
```



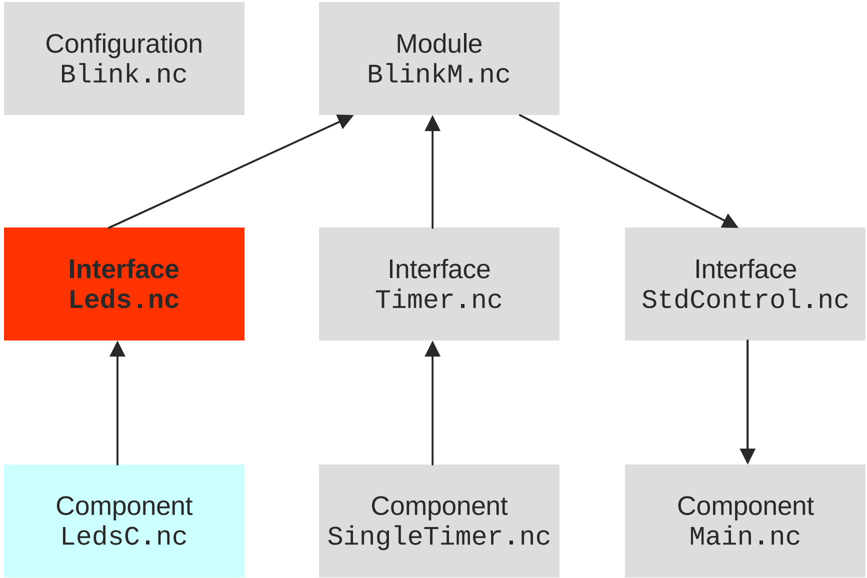
Leds.nc and BlinkM.nc

```
interface Leds {
  command result_t redOn();
  command result_t redOff();
  command result_t redToggle();
  // …
}
module BlinkM { … }
implementation {
  // …
  event result_t Timer.fired()   {
    call Leds.redToggle();
    return SUCCESS;
  }
}
```



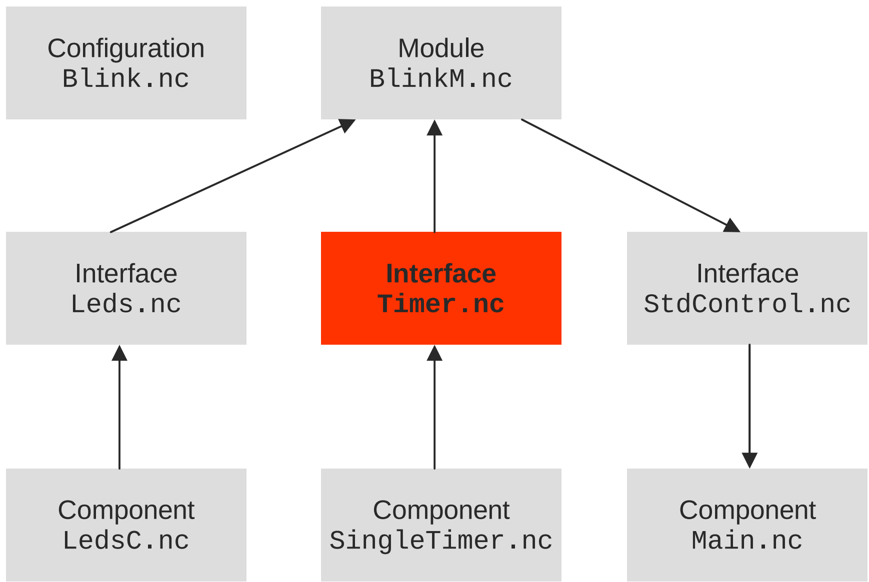
Leds.nc and LedsC.ns

```
interface Leds {
  command result_t redOn();
  command result_t redOff();
  command result_t redToggle();
  // …
}
module LedsC {
  provides interface Leds;
}
implementation {
  command result_t Leds.redOn()
  {
    // …
  }
  // …
}
```



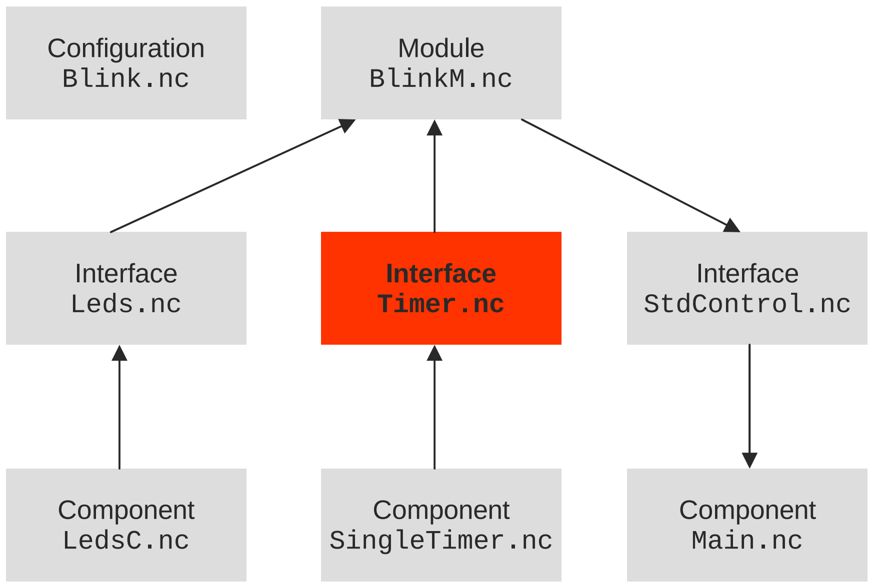
Timer.nc

```
interface Timer {
  command result_t start(char,
        uint32_t);
  command result_t stop();
  event result_t fired();
}
```



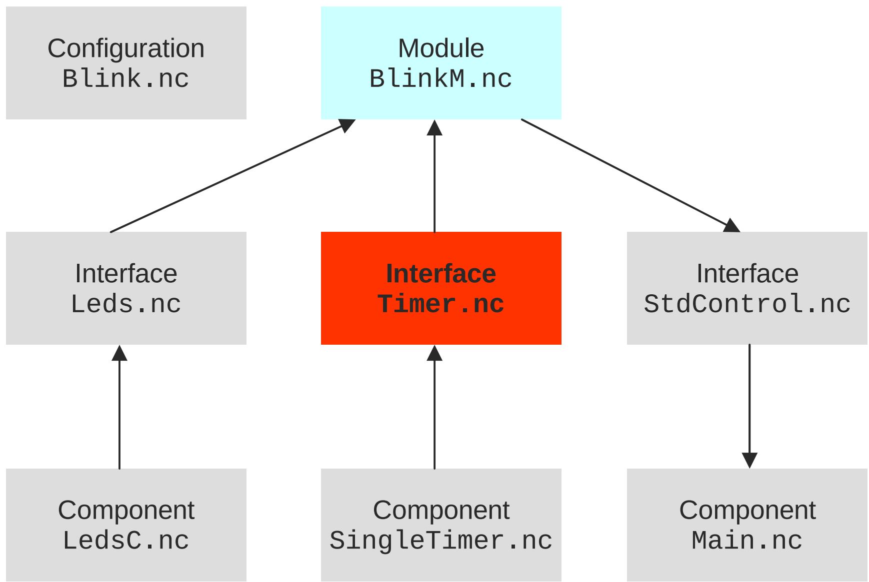
Timer.nc and BlinkM.nc

```
interface Timer {
  command result_t start(char,
      uint32_t);
  command result_t stop();
  event result_t fired();
}
module BlinkM { … }
implementation {
  command result_t StdControl.start()
    call Timer.start(TIMER_REPEAT,
      1000);
    return SUCCESS;
  }
}
```



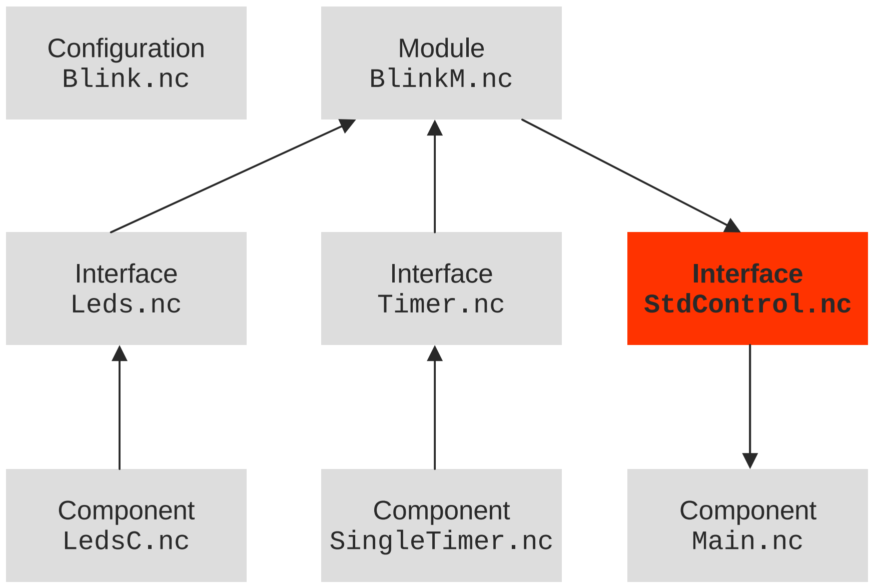
Timer.nc and BlinkM.nc

```
interface Timer {
  command result_t start(char,
       uint32_t);
  command result_t stop();
  event result_t fired();
}
module BlinkM { … }
implementation {
  event result_t Timer.fired()
  {    // …
  }
  // …
}
```



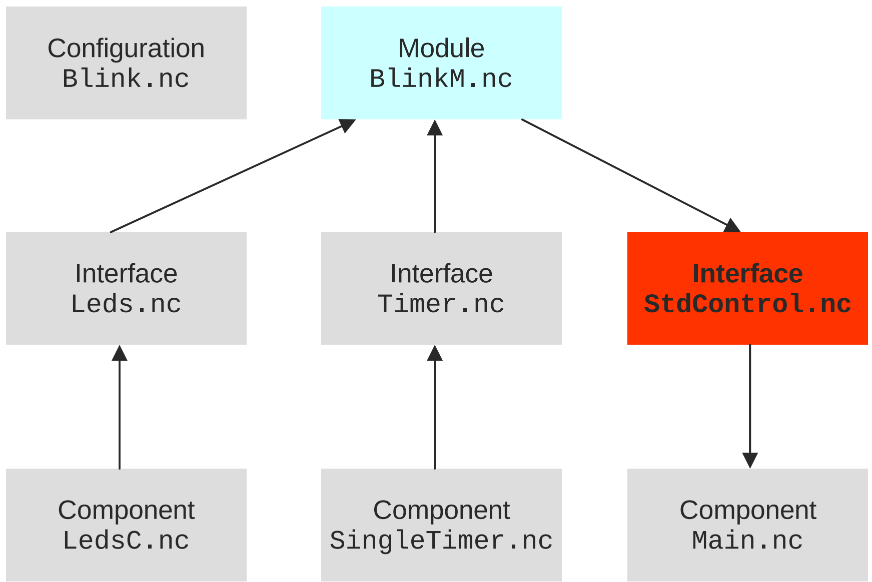
StdControl.nc

```
interface StdControl {
  command result_t init();
  command result_t start();
  command result_t stop();
}
```



StdControl.nc and BlinkM.nc

```
interface StdControl {
  command result_t init();
  command result_t start();
  command result_t stop();
}

module BlinkM {
}
implementation {
  command result_t StdControl.init() {
    return SUCCESS;
  }
  // …
}
```



StdControl.nc and Main.nc

```
interface StdControl {
  command result_t init();
  command result_t start();
  command result_t stop();
}

configuration Main {
  uses interface StdControl;
}
implementation {
  // …
}
```