The Network Layer

Antonio Carzaniga

Faculty of Informatics
University of Lugano

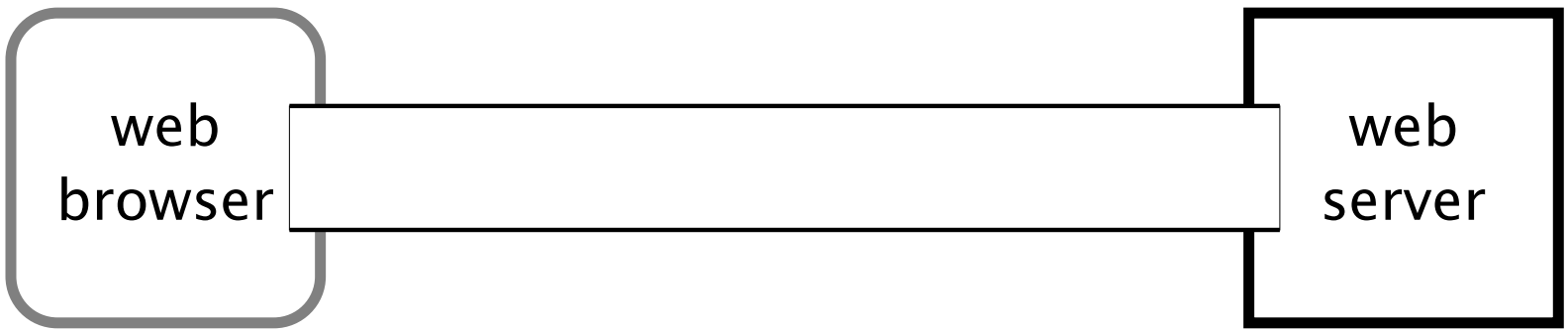
November 26, 2014

Outline

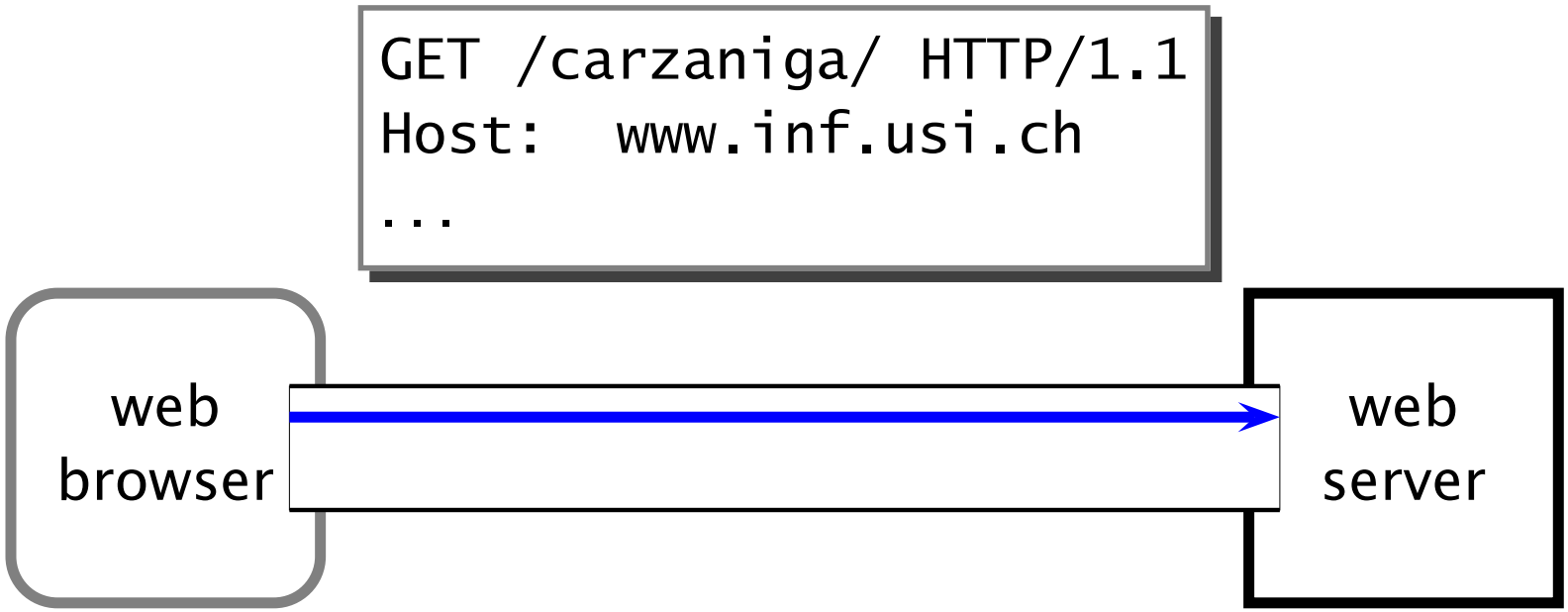
- Basic network-layer architecture of a datagram network
- Introduction to forwarding
- Introduction to routing
- General architecture of a router
- Switching fabric and queuing
- Internet network-layer protocol
- The Internet protocol (IP)
- Fragmentation

Application Level

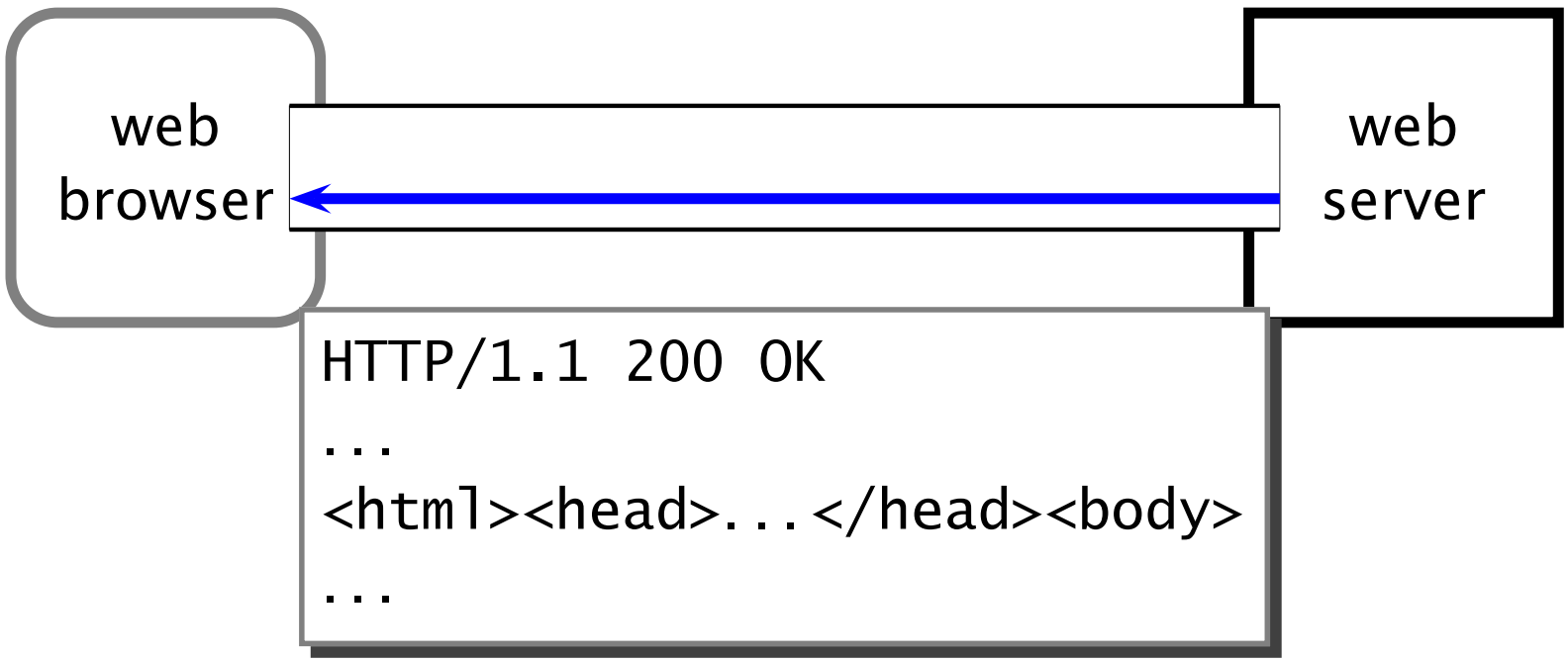
Application Level



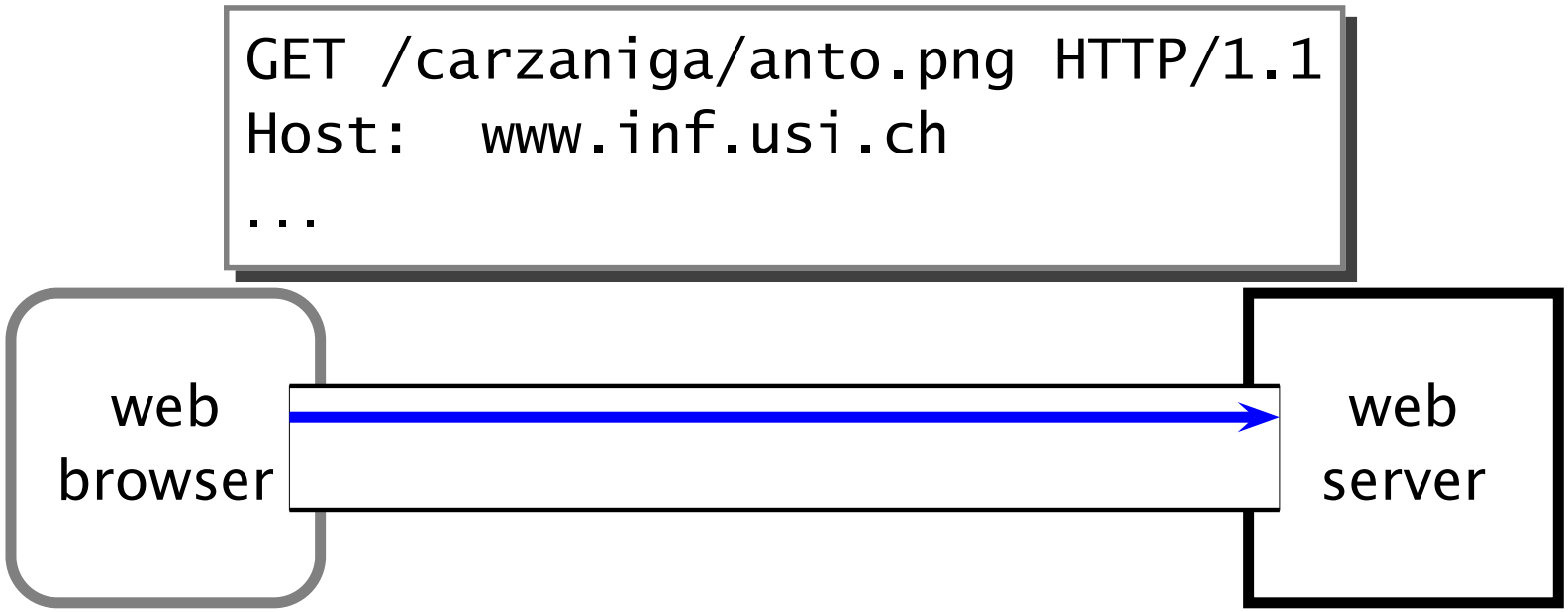
Application Level



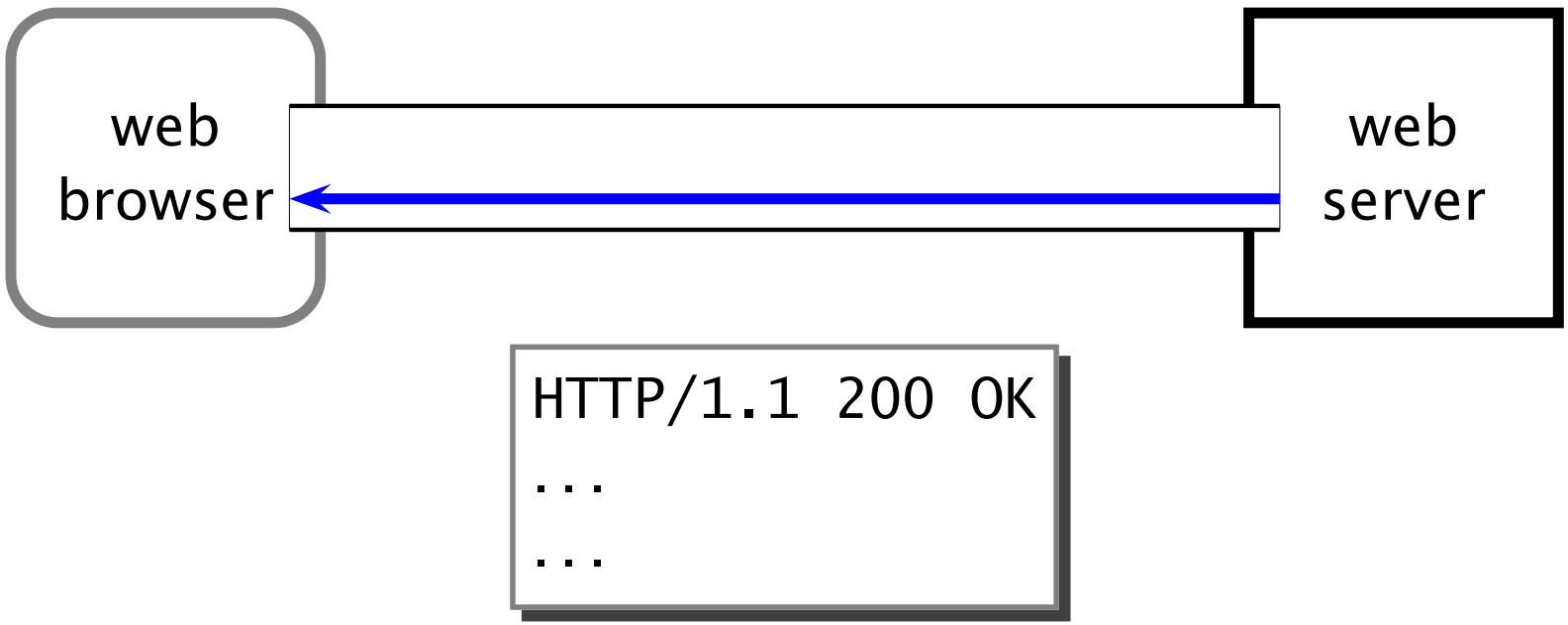
Application Level



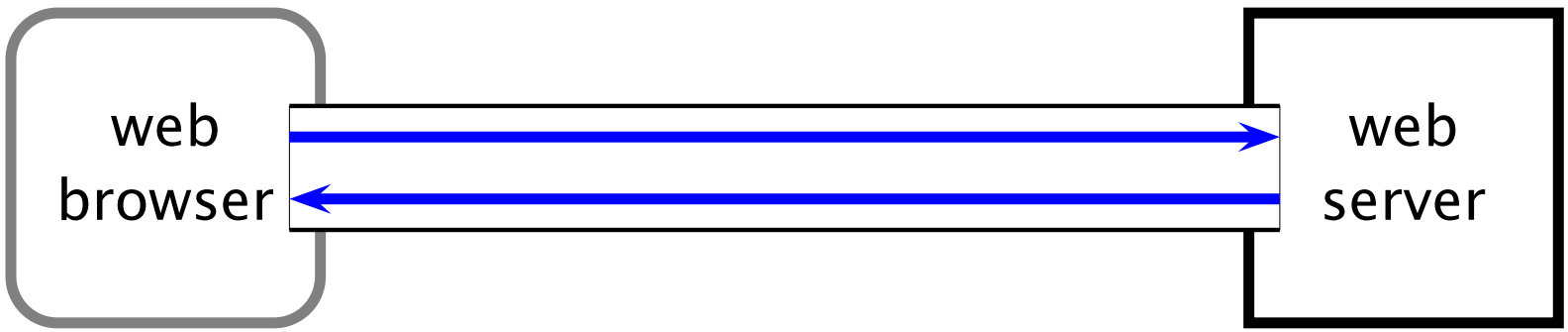
Application Level



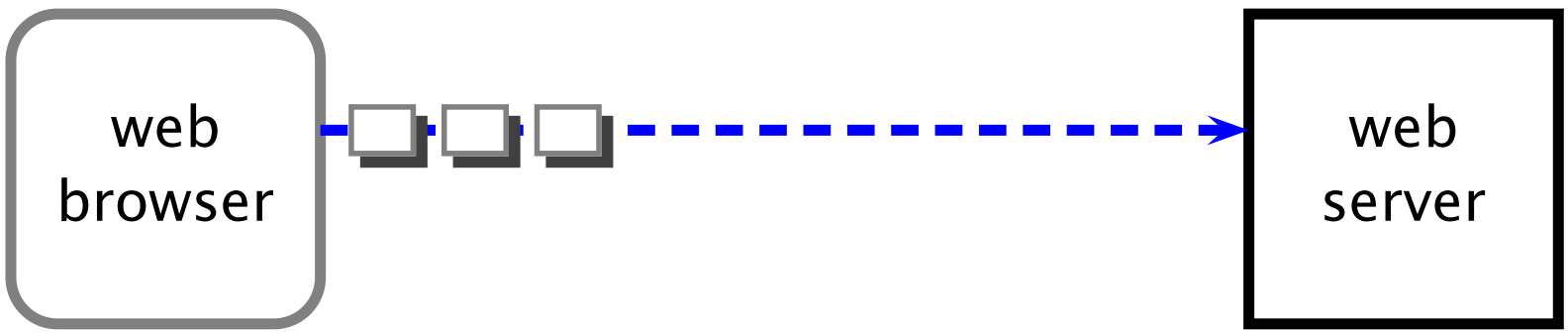
Application Level



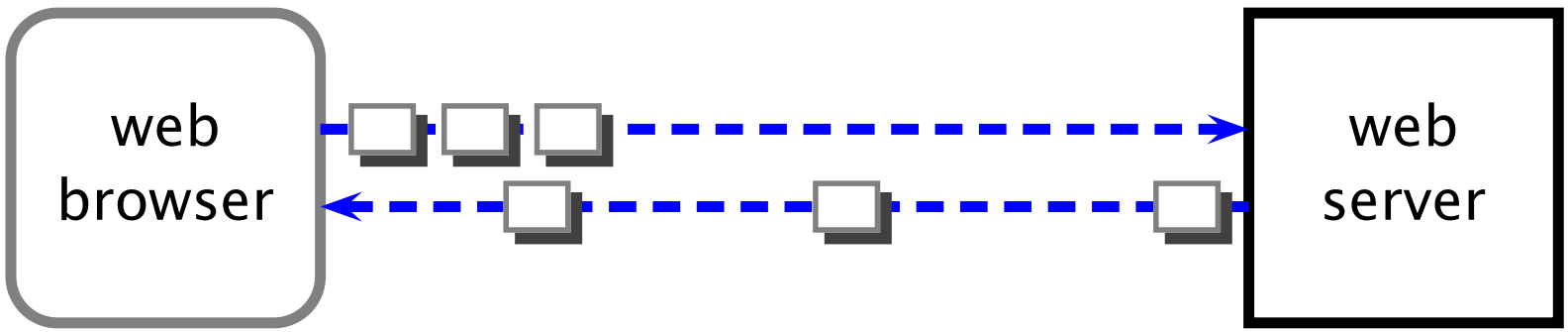
Transport Level



Transport Level

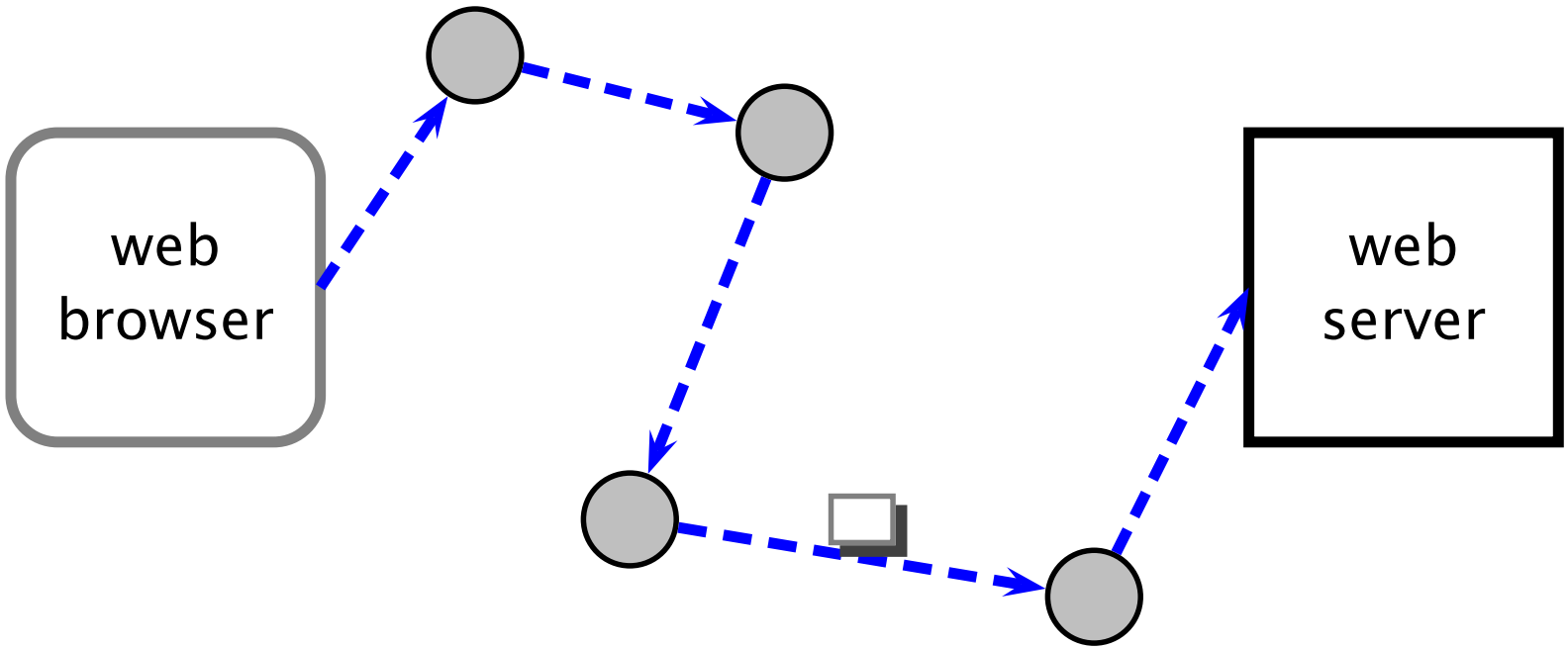


Transport Level

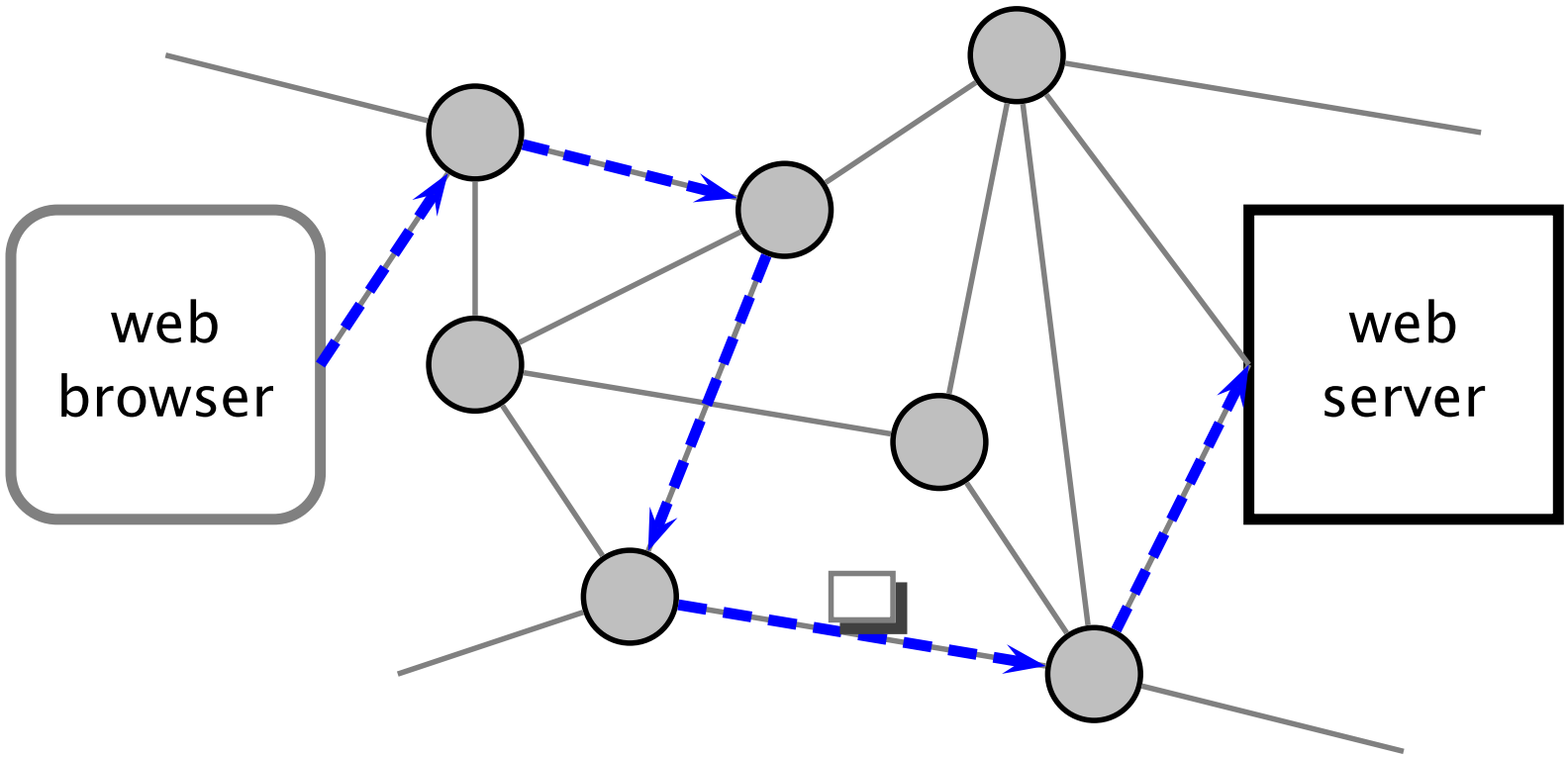


Network Layer

Network Layer

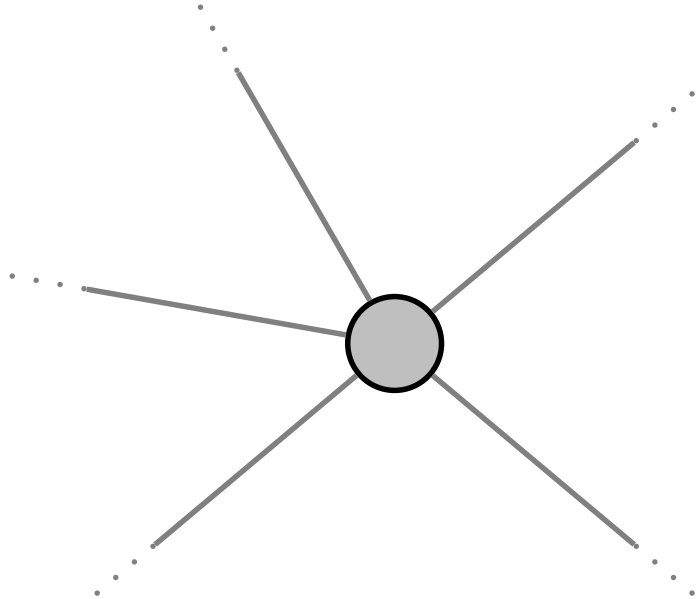


Network Layer



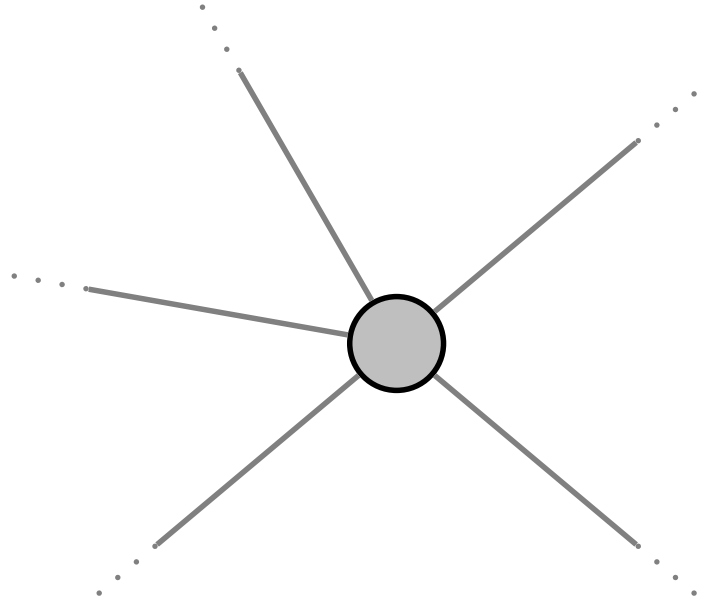
Router

Router



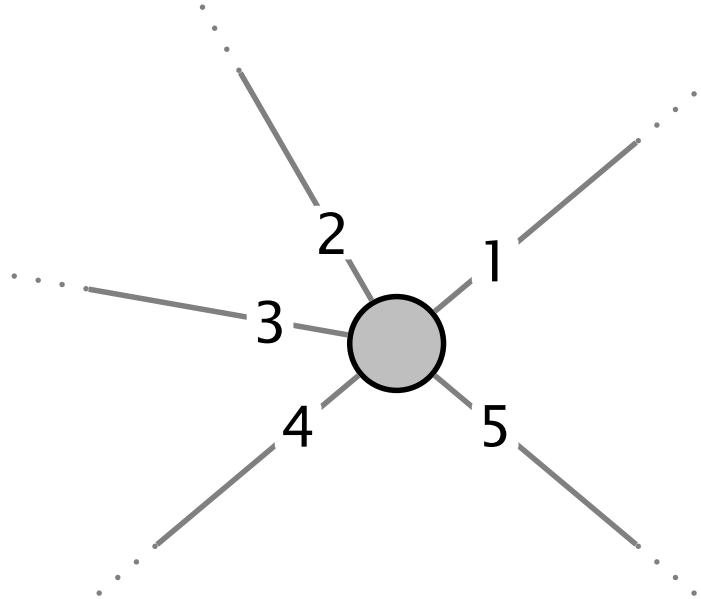
- Fundamental component of the network layer

Router



- Fundamental component of the network layer
- A *node in a graph*

Router



- Fundamental component of the network layer
- A *node in a graph*
- A finite set of input/output (physical) connections
  - a.k.a., *interfaces* or *ports*

Focus: “Datagram” Networks

Focus: "Datagram" Networks

- *Packet-switched network*

Focus: "Datagram" Networks

- *Packet-switched network*
  - information is transmitted in discrete units called *datagrams*

Focus: "Datagram" Networks

- *Packet-switched network*
  - information is transmitted in discrete units called *datagrams*
- *Connectionless service*

Focus: "Datagram" Networks

- *Packet-switched network*
  - information is transmitted in discrete units called *datagrams*
- *Connectionless service*
  - a datagram is a *self-contained message*
  - treated independently by the network
  - no connection setup/tear-down phase

Focus: "Datagram" Networks

- *Packet-switched network*
  - information is transmitted in discrete units called *datagrams*
- *Connectionless service*
  - a datagram is a *self-contained message*
  - treated independently by the network
  - no connection setup/tear-down phase
- *"Best-effort" service*

Focus: "Datagram" Networks

- *Packet-switched network*
  - information is transmitted in discrete units called *datagrams*
- *Connectionless service*
  - a datagram is a *self-contained message*
  - treated independently by the network
  - no connection setup/tear-down phase
- *"Best-effort" service*
  - delivery guarantee: none

Focus: "Datagram" Networks

- *Packet-switched network*
  - information is transmitted in discrete units called *datagrams*
- *Connectionless service*
  - a datagram is a *self-contained message*
  - treated independently by the network
  - no connection setup/tear-down phase
- *"Best-effort" service*
  - delivery guarantee: none
  - maximum latency guarantee: none

Focus: "Datagram" Networks

- *Packet-switched network*
  - information is transmitted in discrete units called *datagrams*
- *Connectionless service*
  - a datagram is a *self-contained message*
  - treated independently by the network
  - no connection setup/tear-down phase
- *"Best-effort" service*
  - delivery guarantee: none
  - maximum latency guarantee: none
  - bandwidth guarantee: none

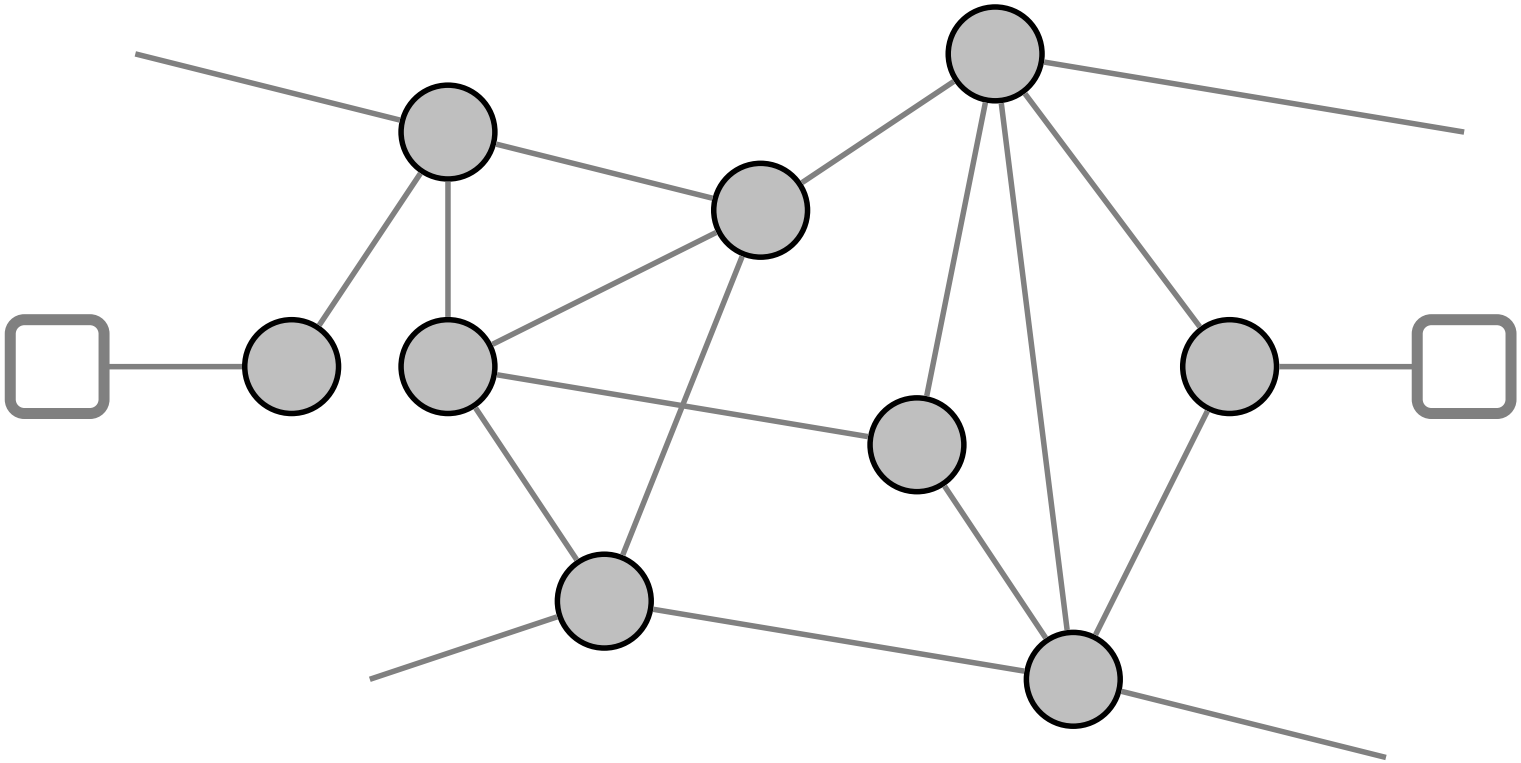
Focus: "Datagram" Networks

- *Packet-switched network*
  - information is transmitted in discrete units called *datagrams*
- *Connectionless service*
  - a datagram is a *self-contained message*
  - treated independently by the network
  - no connection setup/tear-down phase
- *"Best-effort" service*
  - delivery guarantee: none
  - maximum latency guarantee: none
  - bandwidth guarantee: none
  - in-order delivery guarantee: none

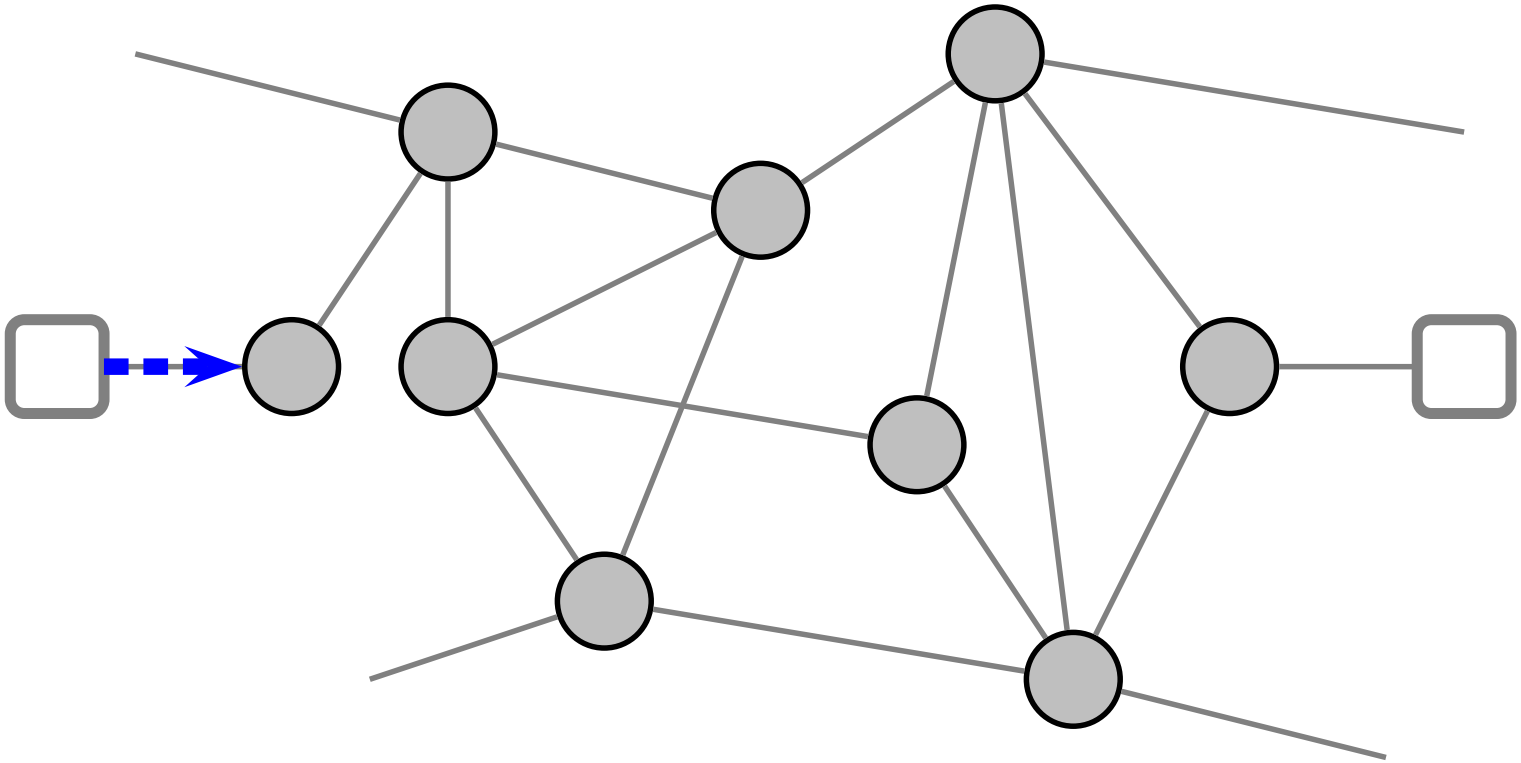
Focus: "Datagram" Networks

- *Packet-switched network*
  - information is transmitted in discrete units called *datagrams*
- *Connectionless service*
  - a datagram is a *self-contained message*
  - treated independently by the network
  - no connection setup/tear-down phase
- *"Best-effort" service*
  - delivery guarantee: none
  - maximum latency guarantee: none
  - bandwidth guarantee: none
  - in-order delivery guarantee: none
  - congestion indication: none

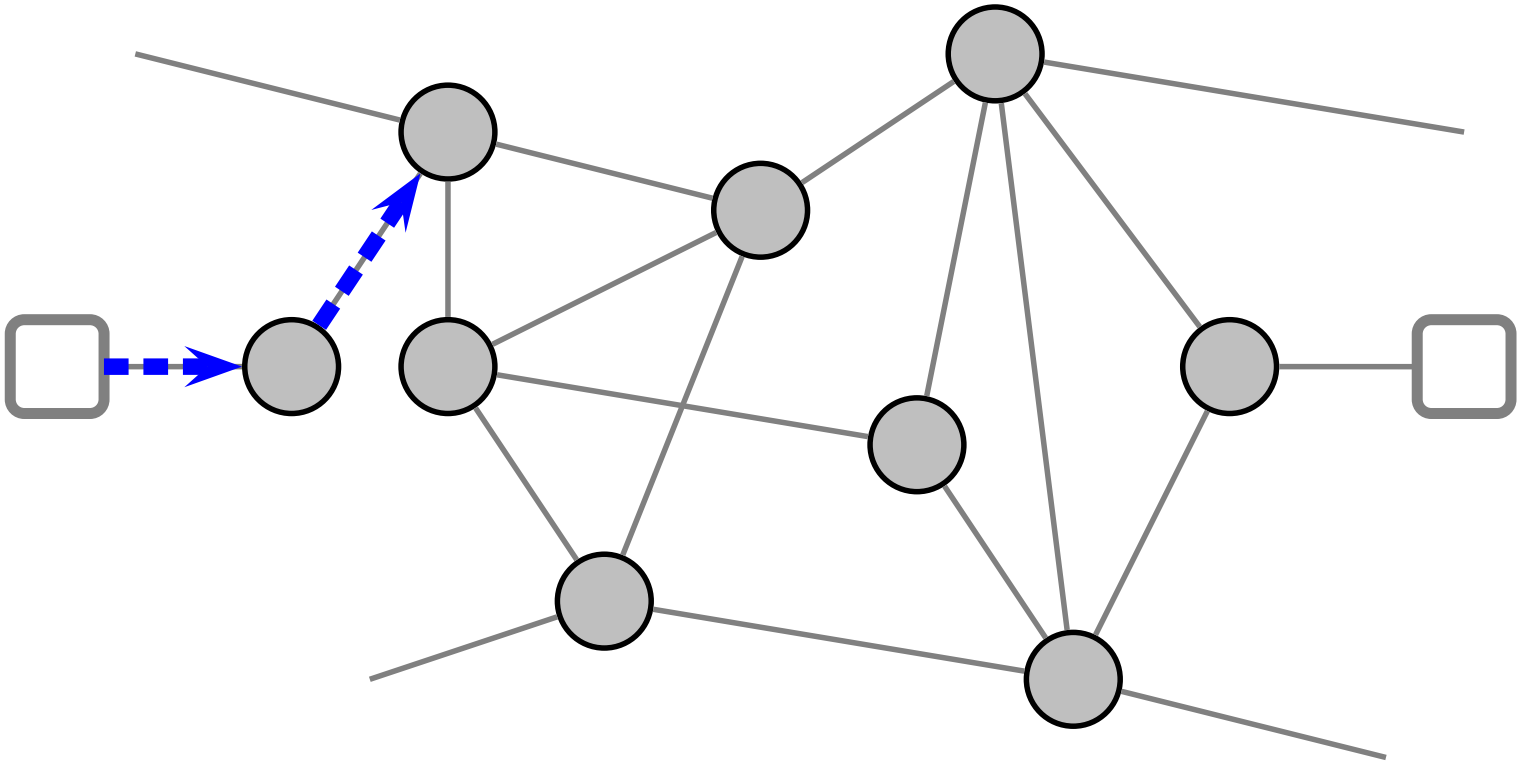
Datagram Network



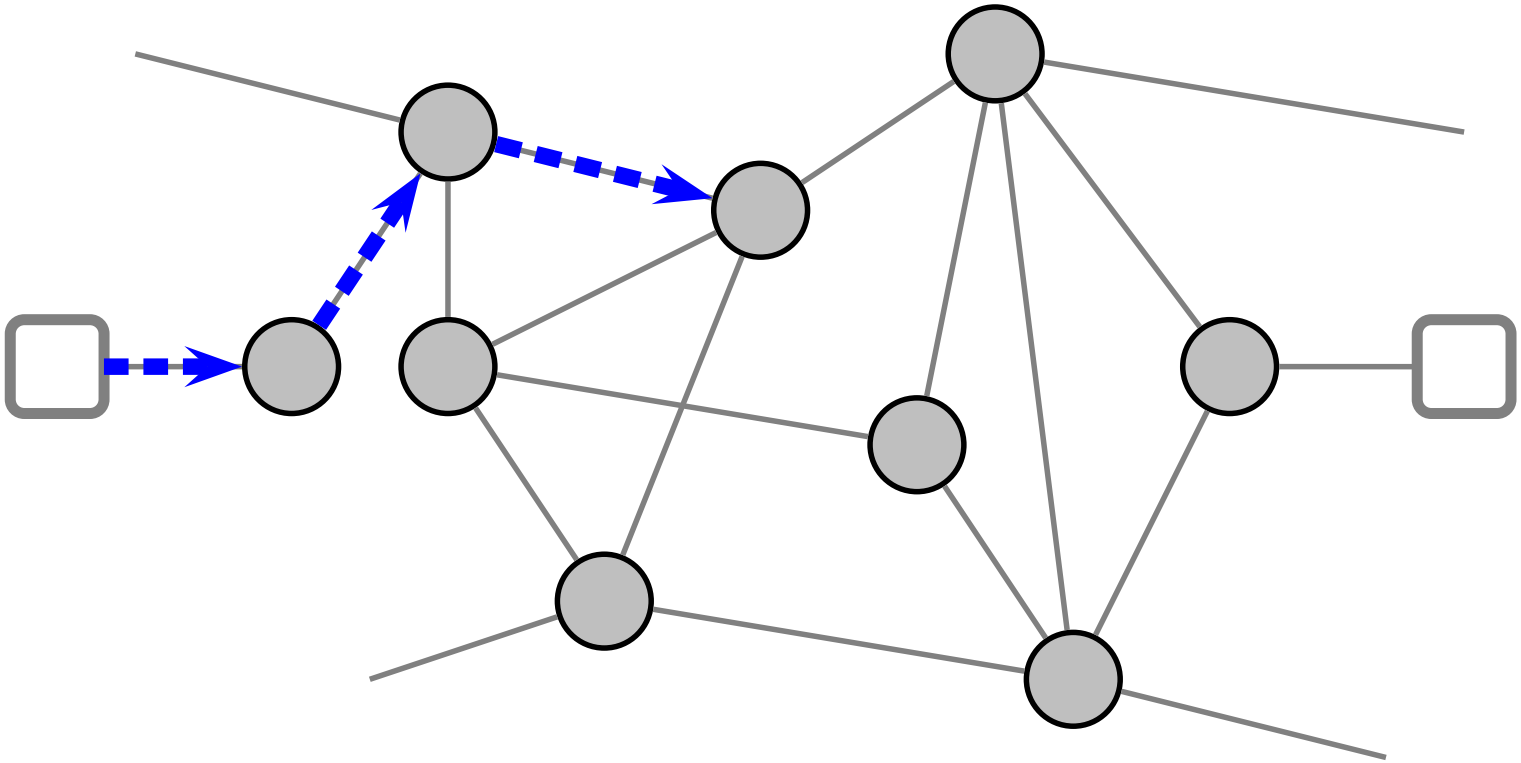
Datagram Network



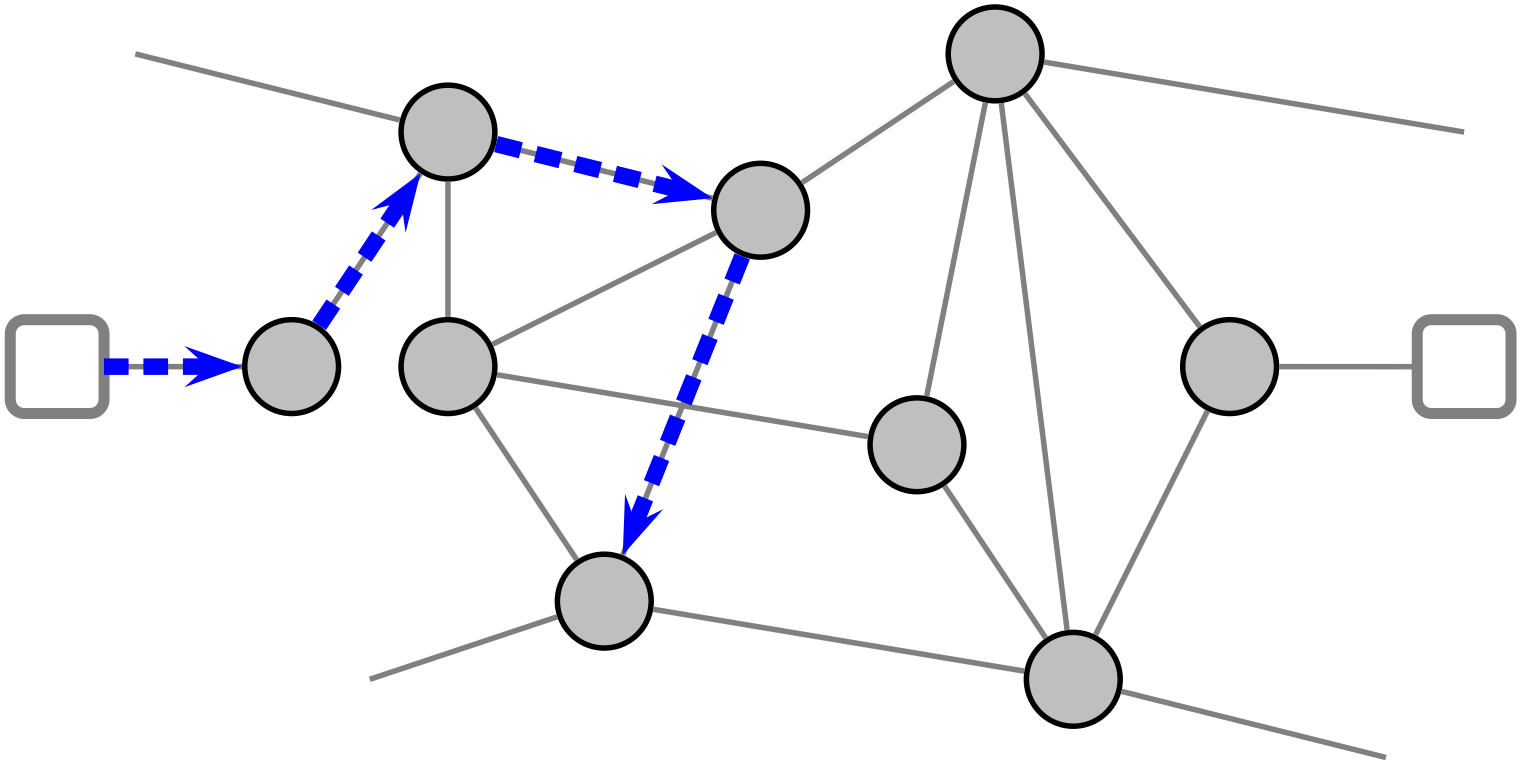
Datagram Network



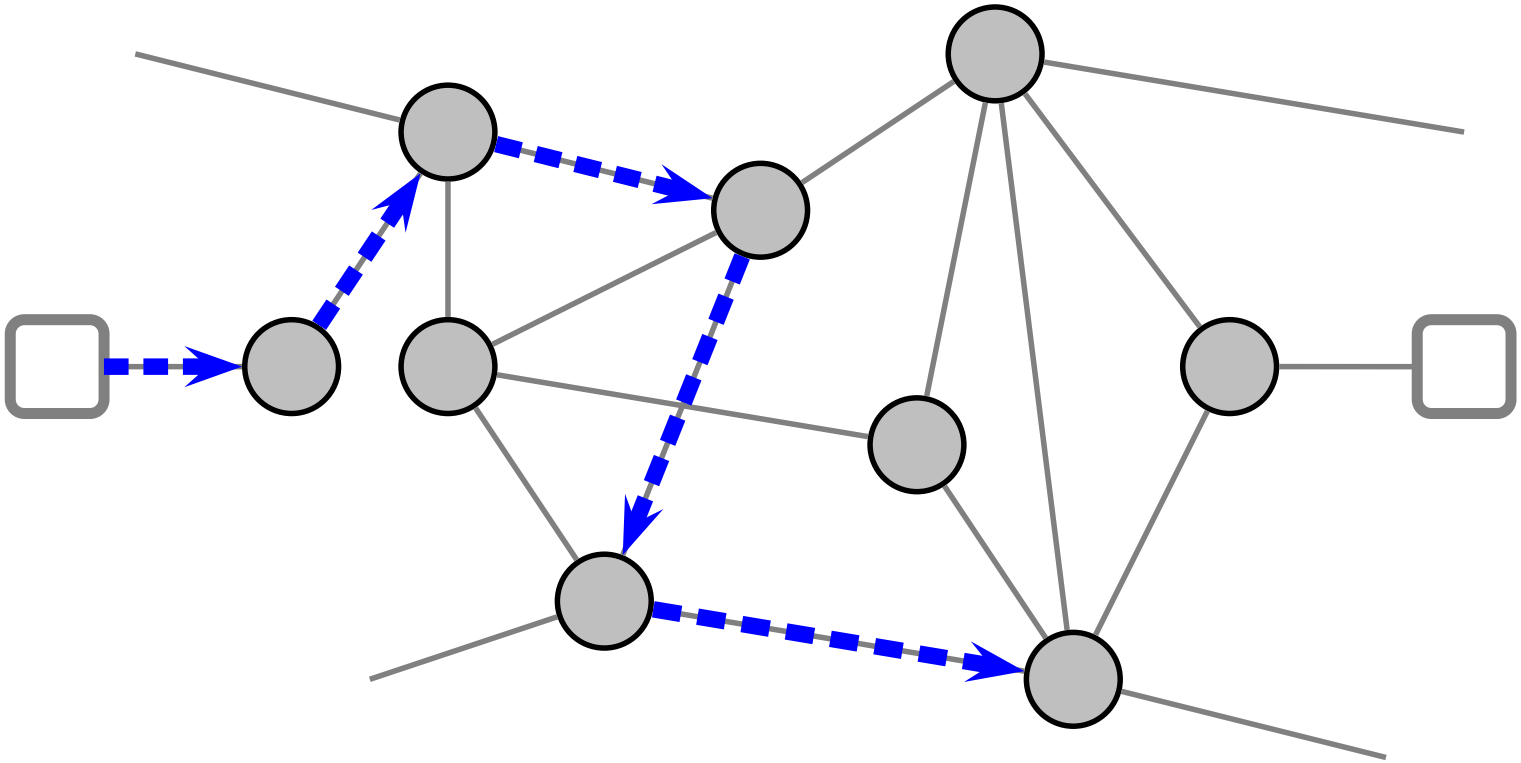
Datagram Network



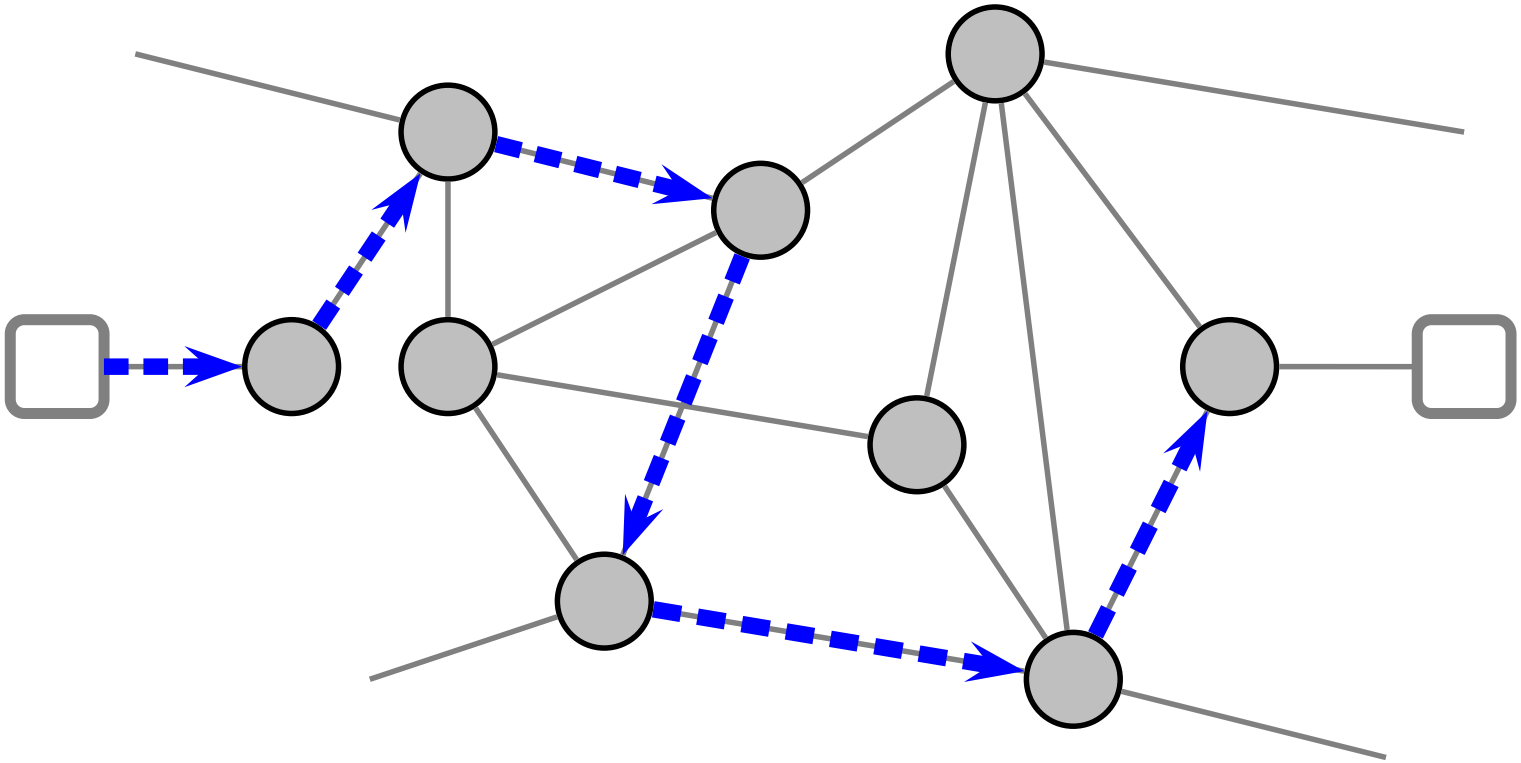
Datagram Network



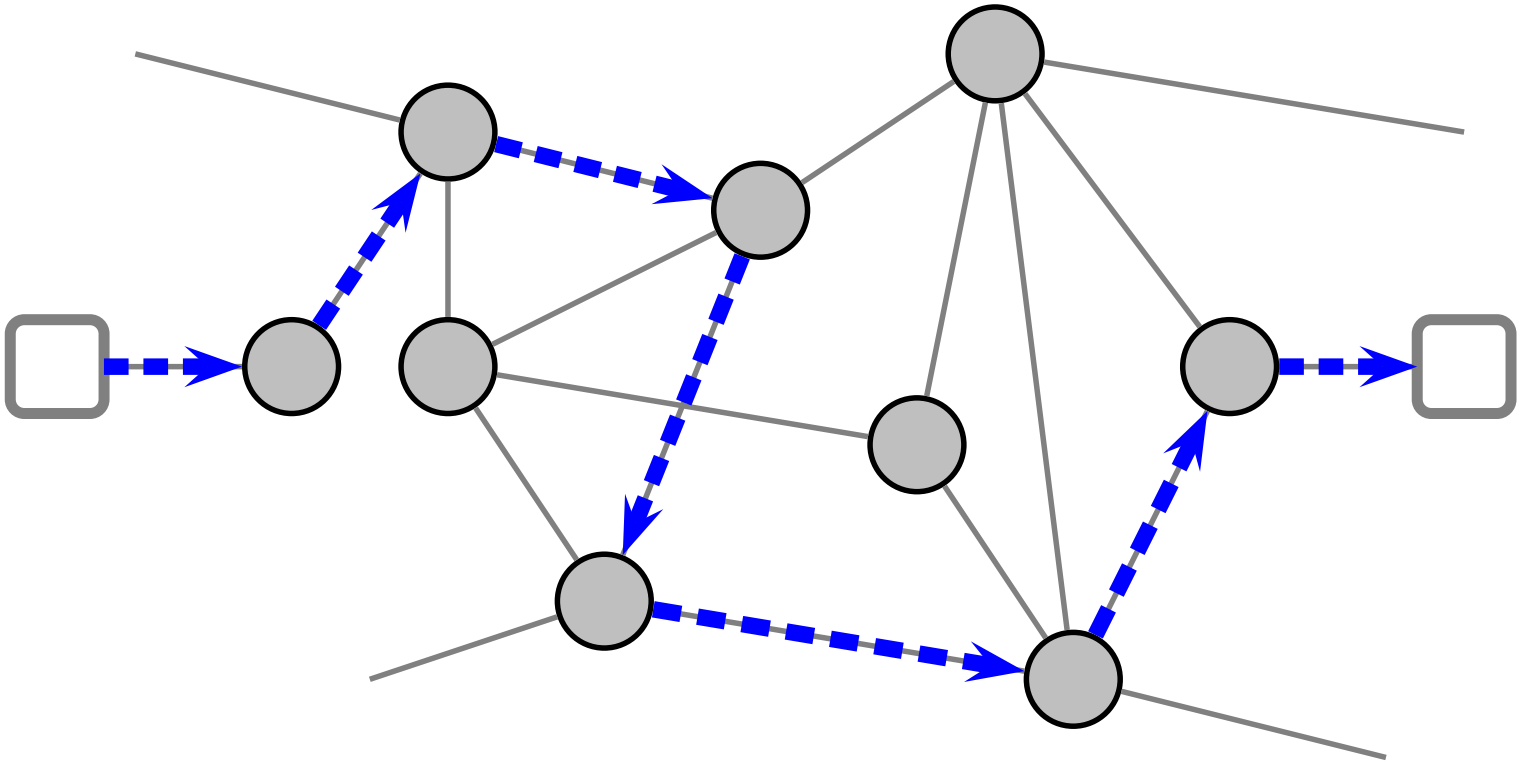
Datagram Network



Datagram Network

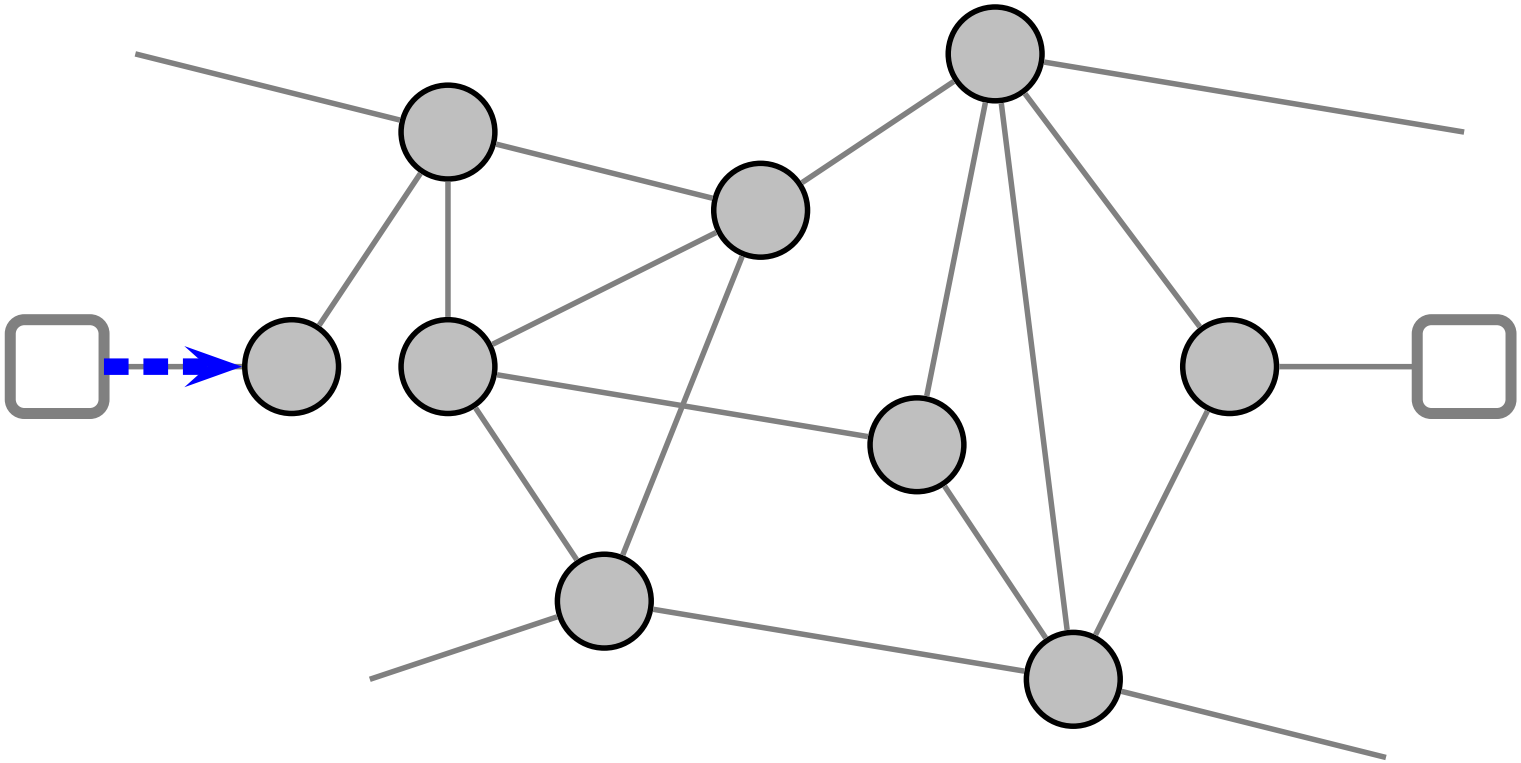


Datagram Network



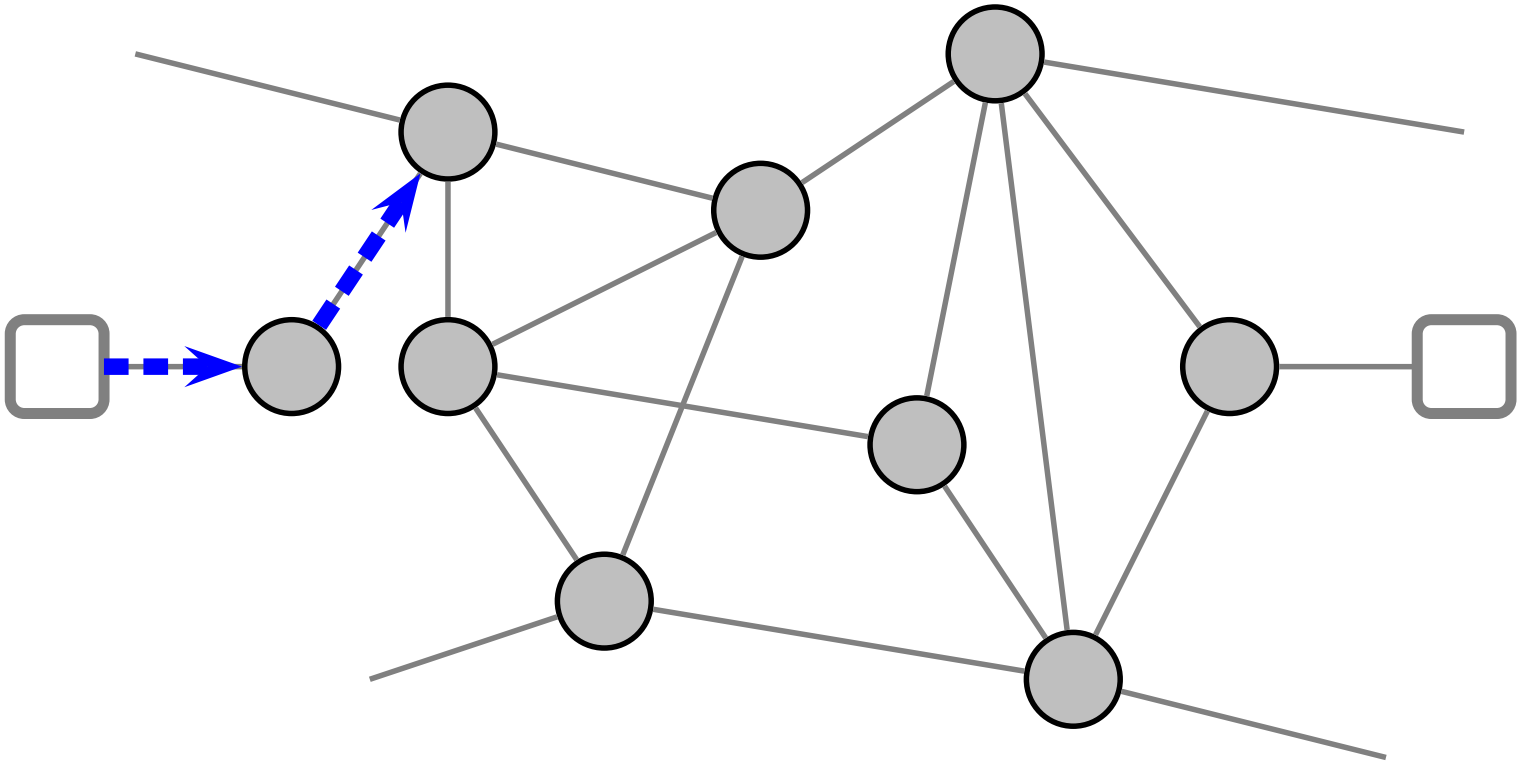
- Potentially *multiple paths* for the same source/destination

Datagram Network



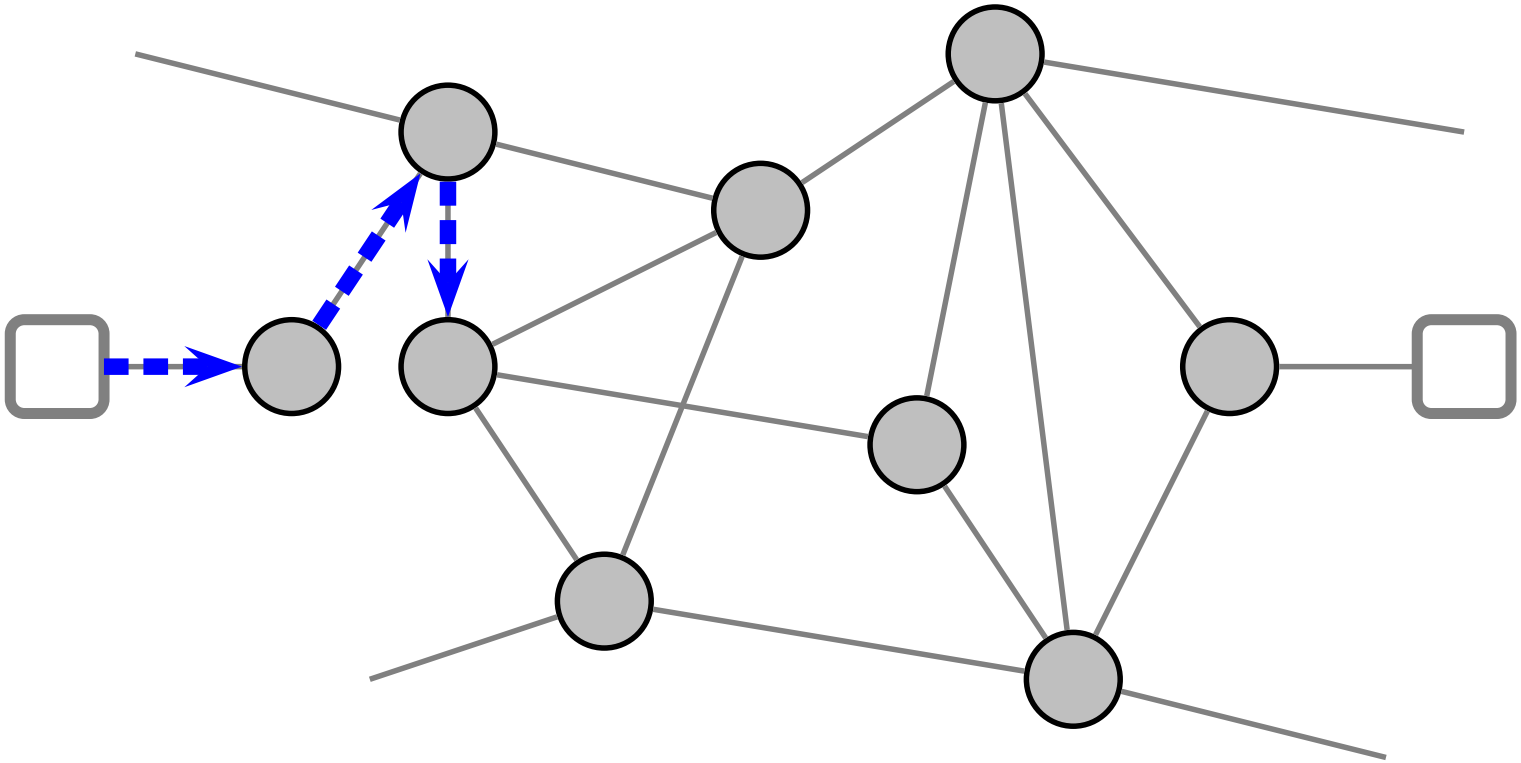
- Potentially *multiple paths* for the same source/destination

Datagram Network



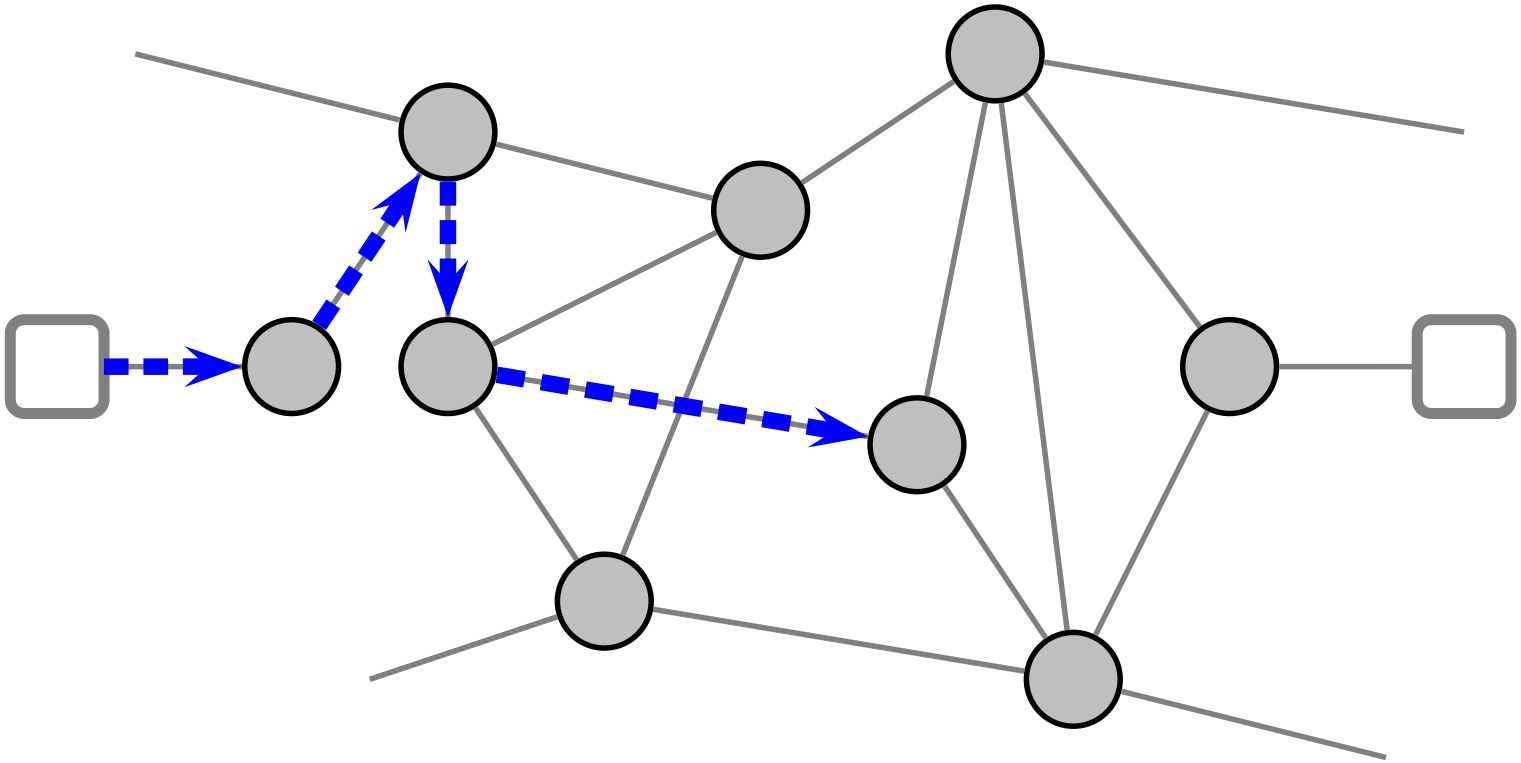
- Potentially *multiple paths* for the same source/destination

Datagram Network



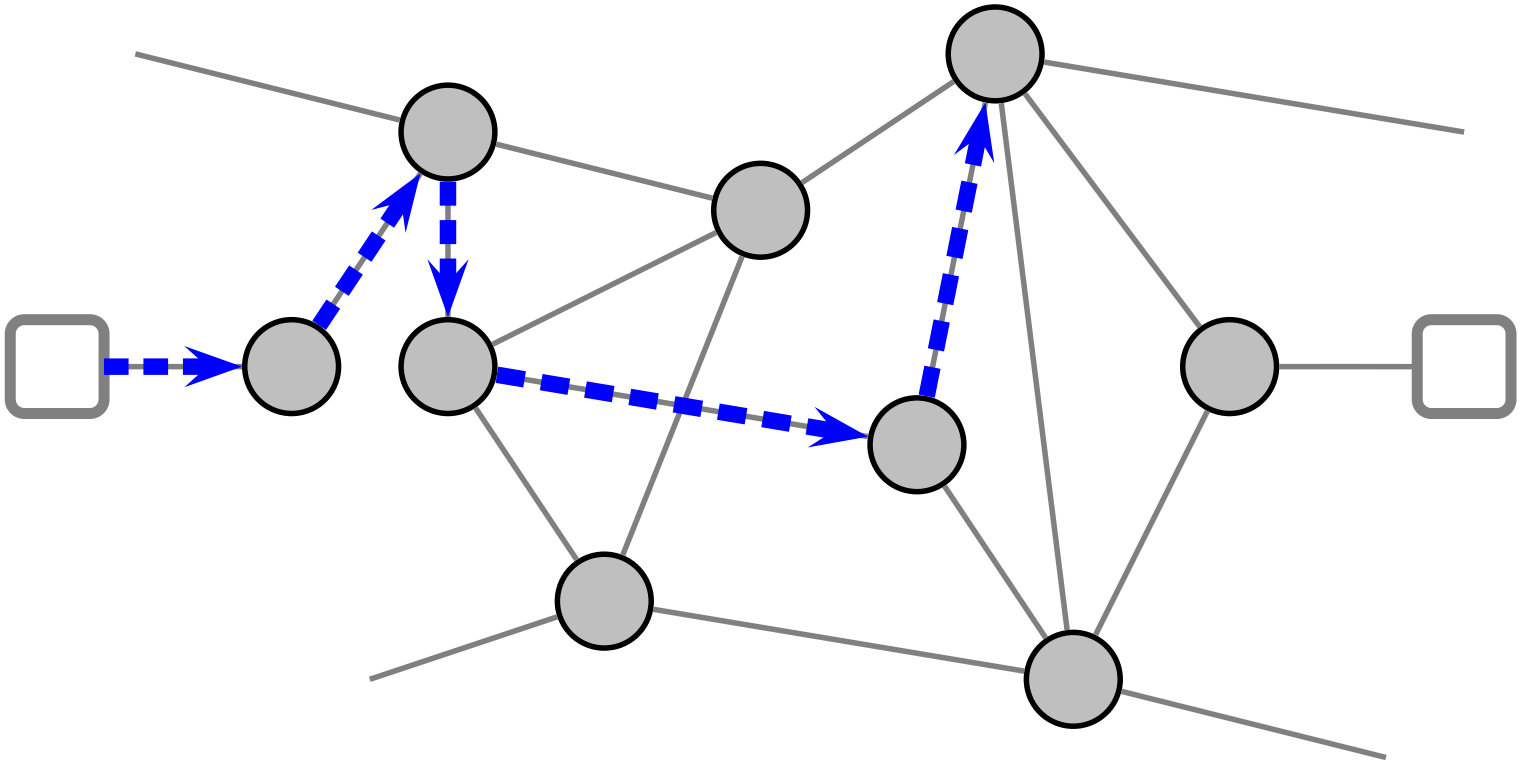
- Potentially *multiple paths* for the same source/destination

Datagram Network



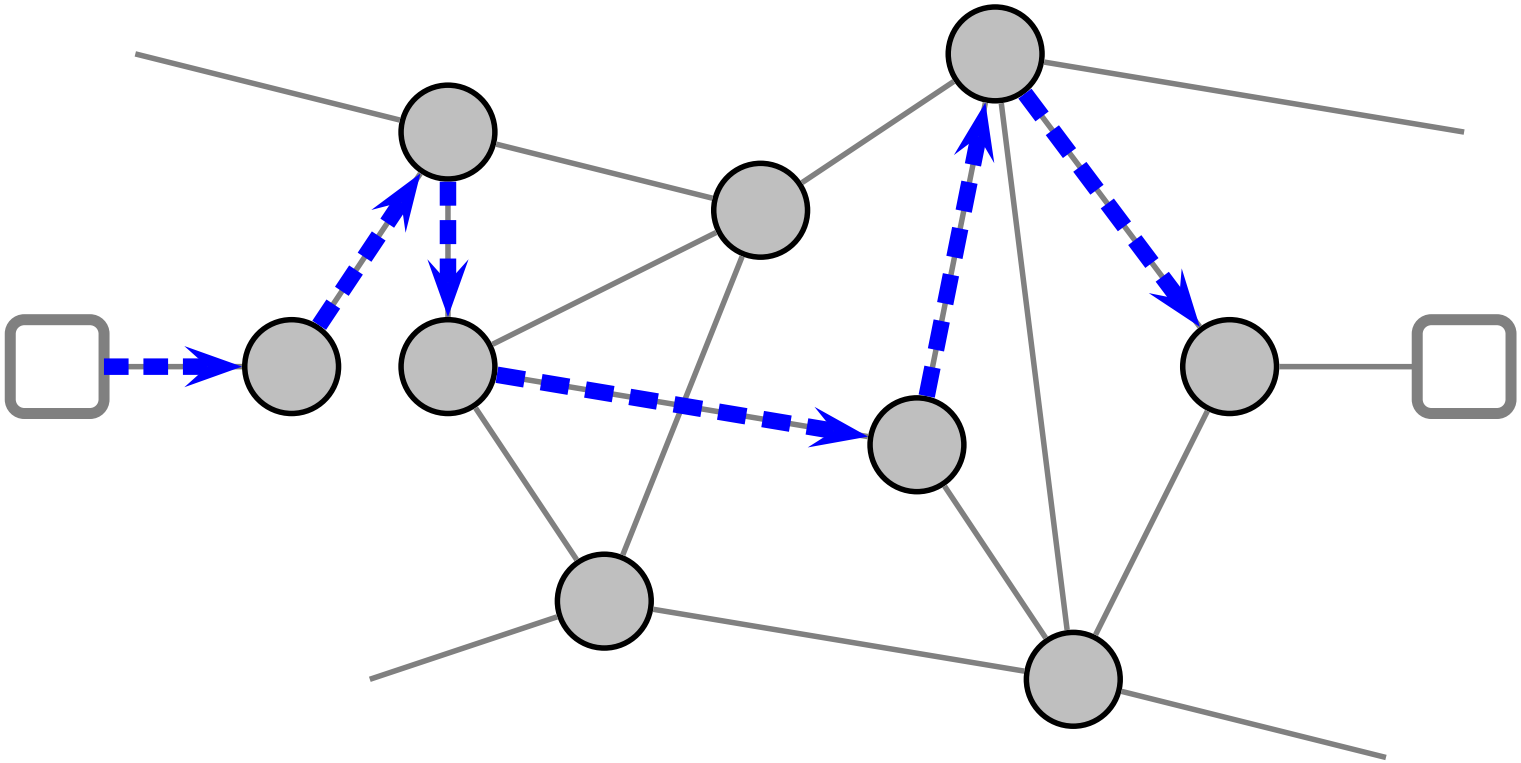
- Potentially *multiple paths* for the same source/destination

Datagram Network



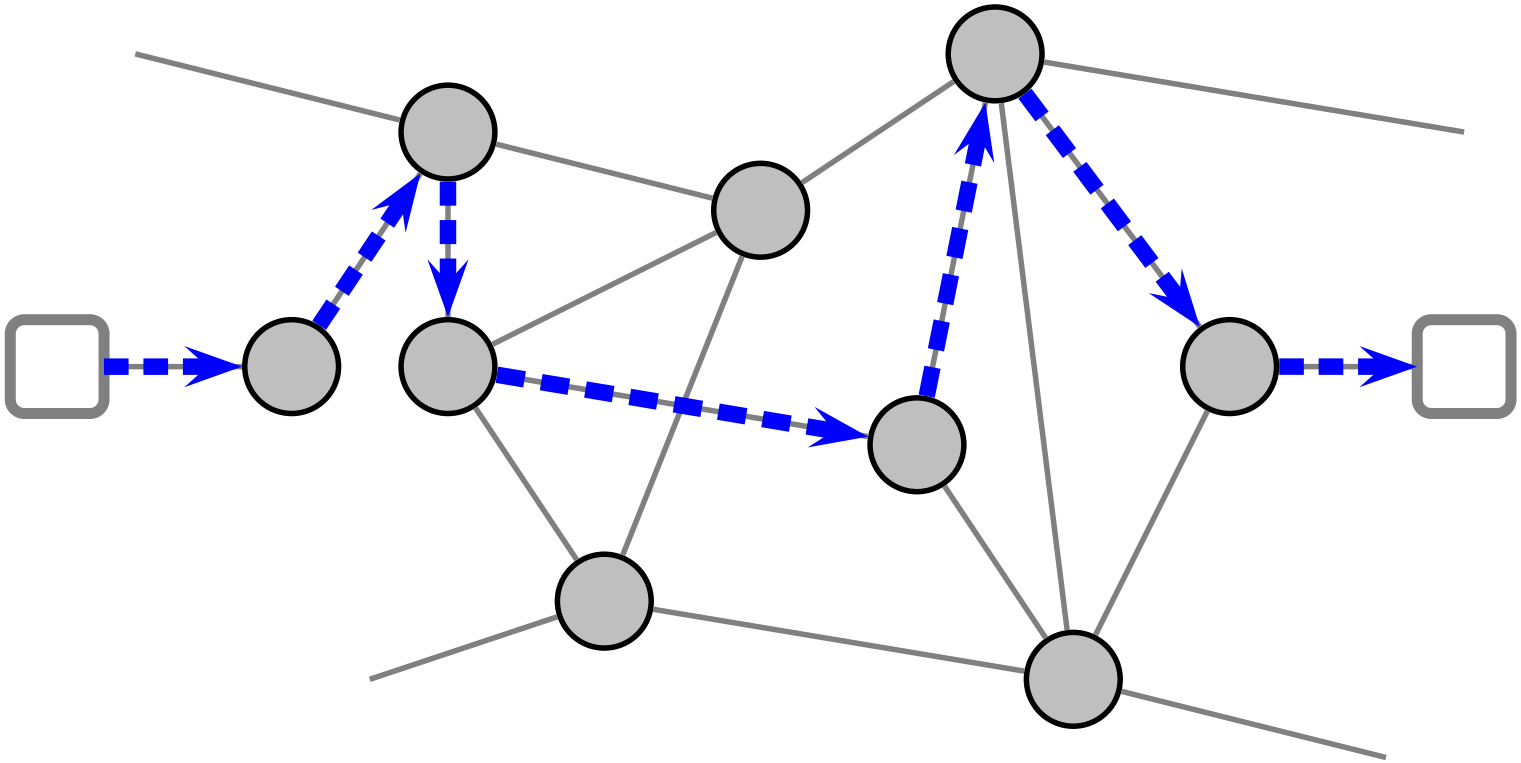
- Potentially *multiple paths* for the same source/destination

Datagram Network



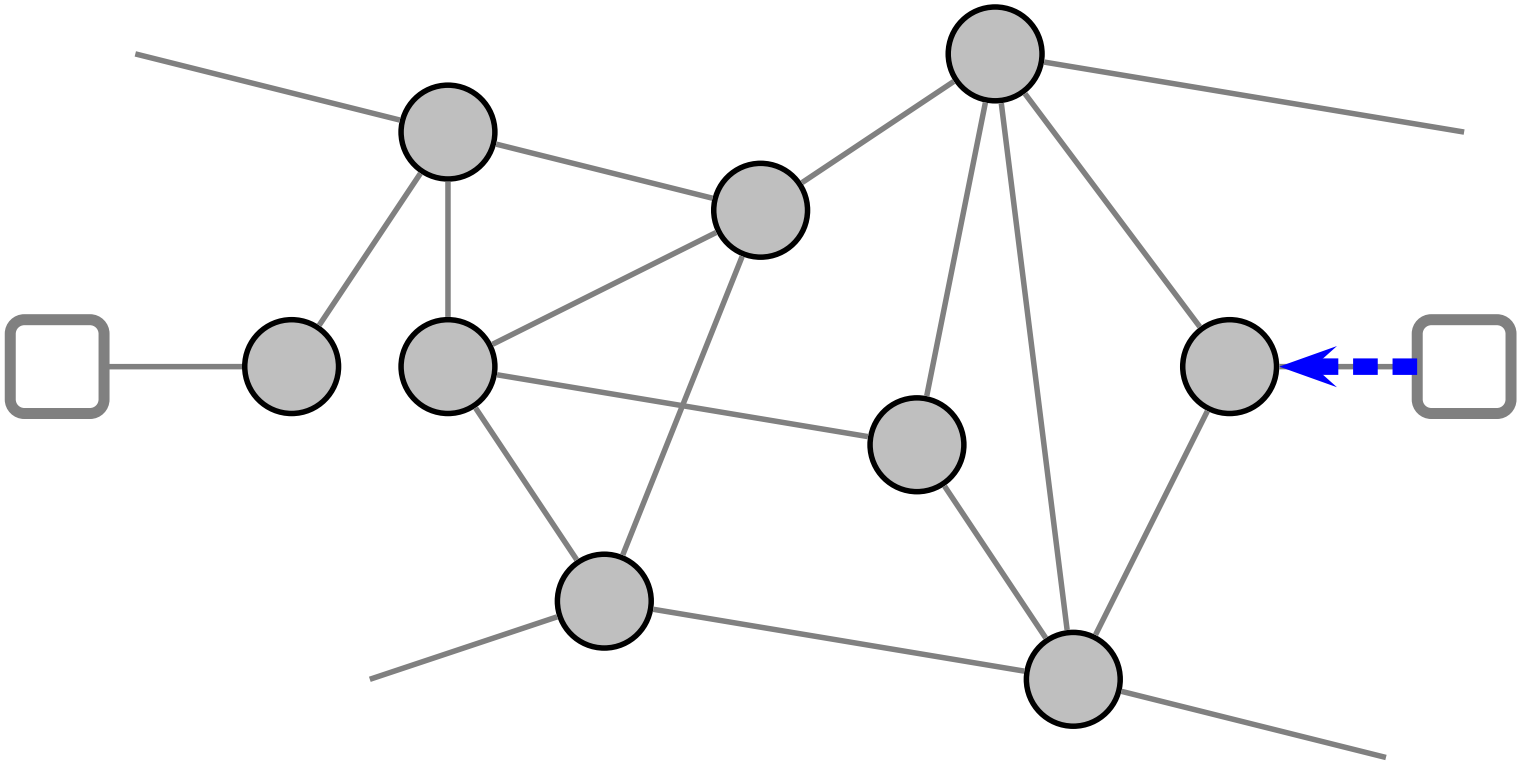
- Potentially *multiple paths* for the same source/destination

Datagram Network



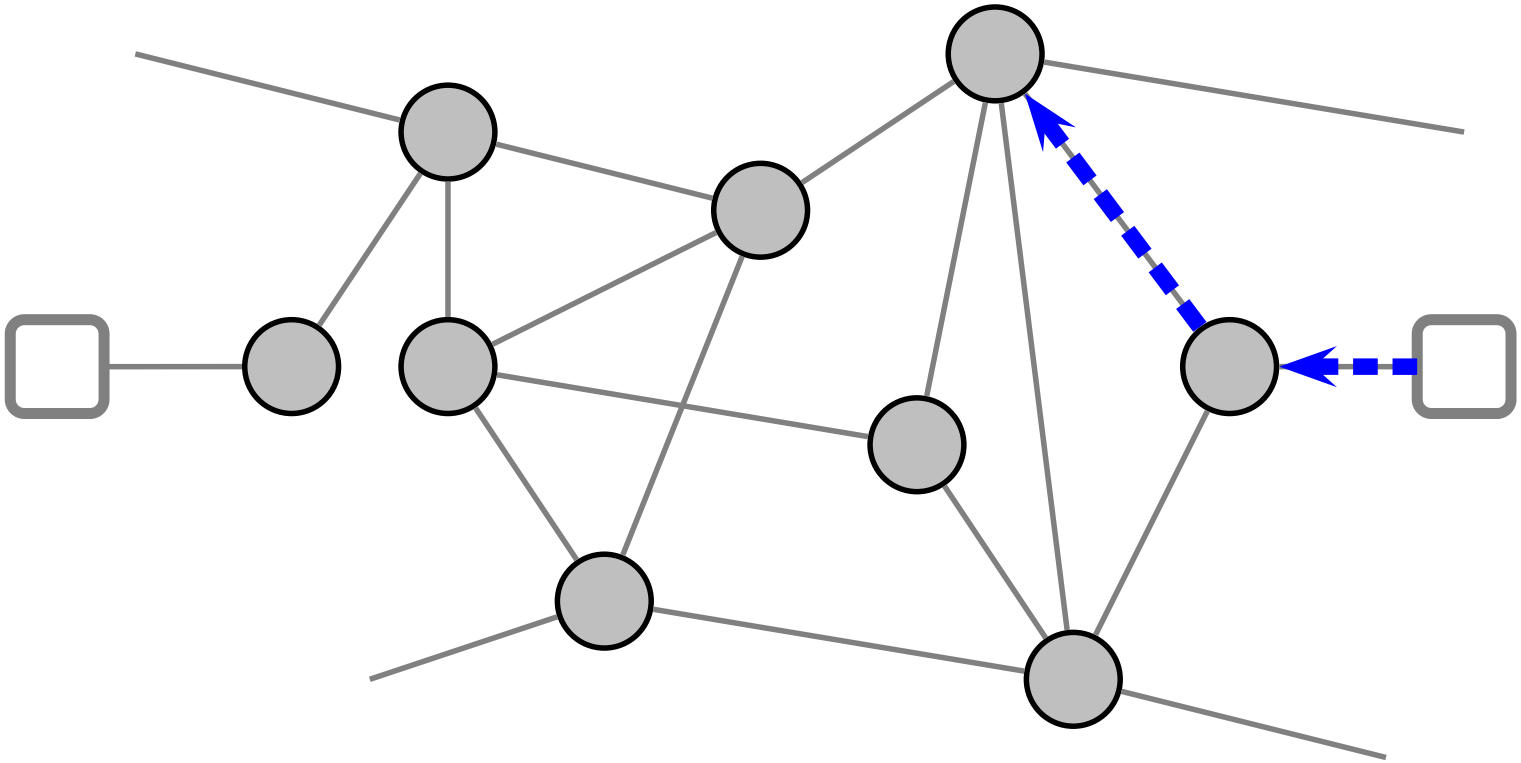
- Potentially *multiple paths* for the same source/destination

Datagram Network



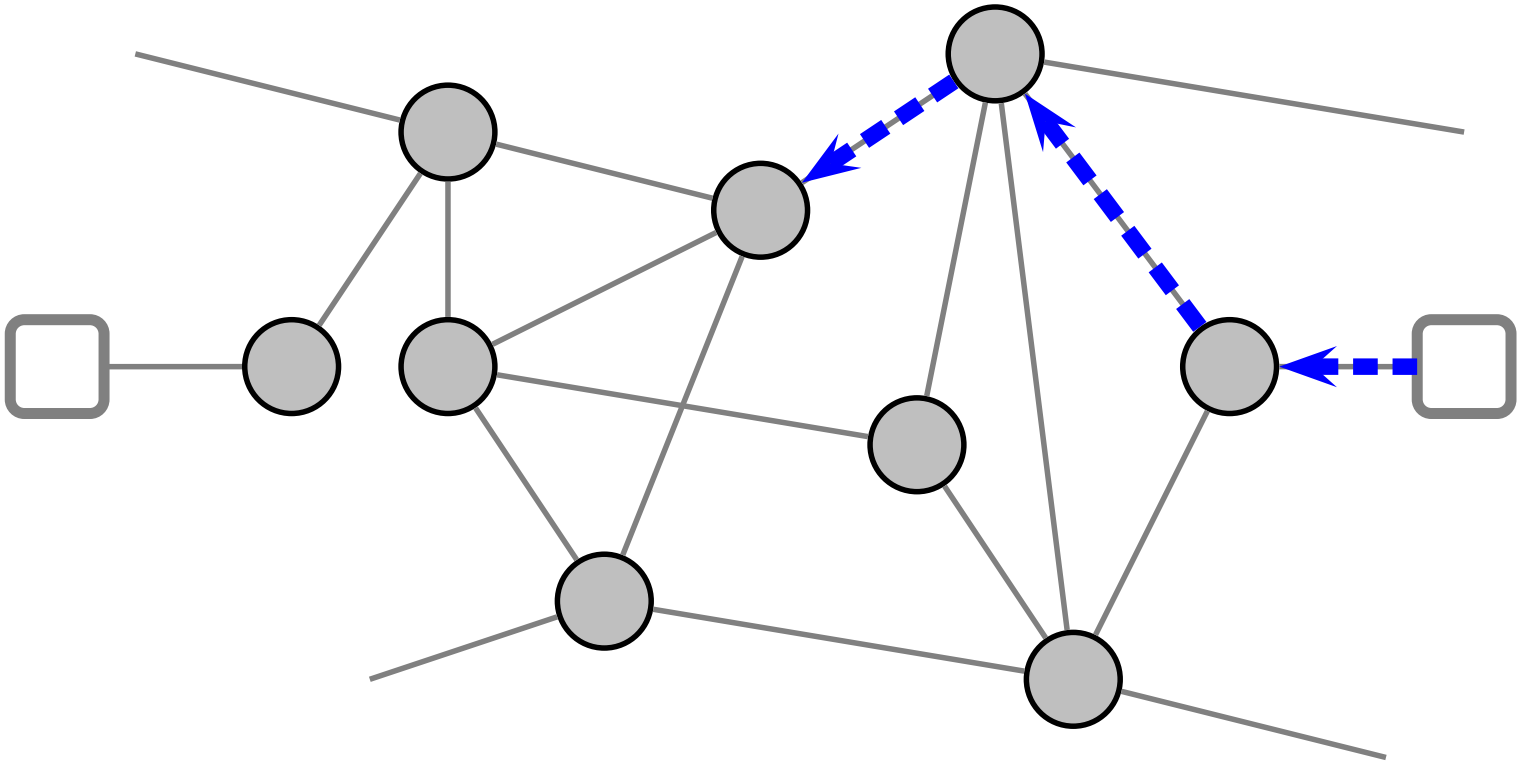
- Potentially *multiple paths* for the same source/destination
- Potentially *asymmetric paths*

Datagram Network



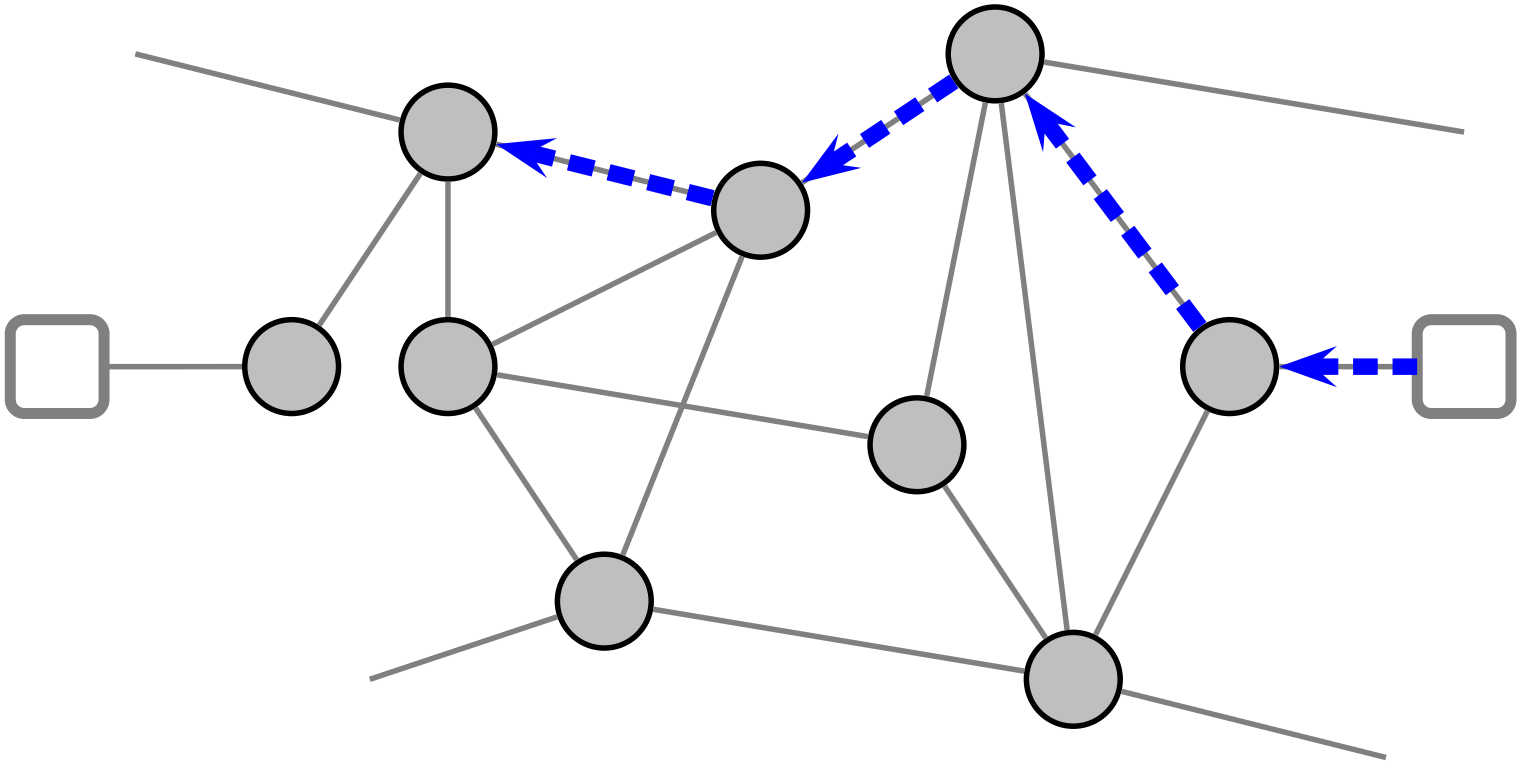
- Potentially *multiple paths* for the same source/destination
- Potentially *asymmetric paths*

Datagram Network



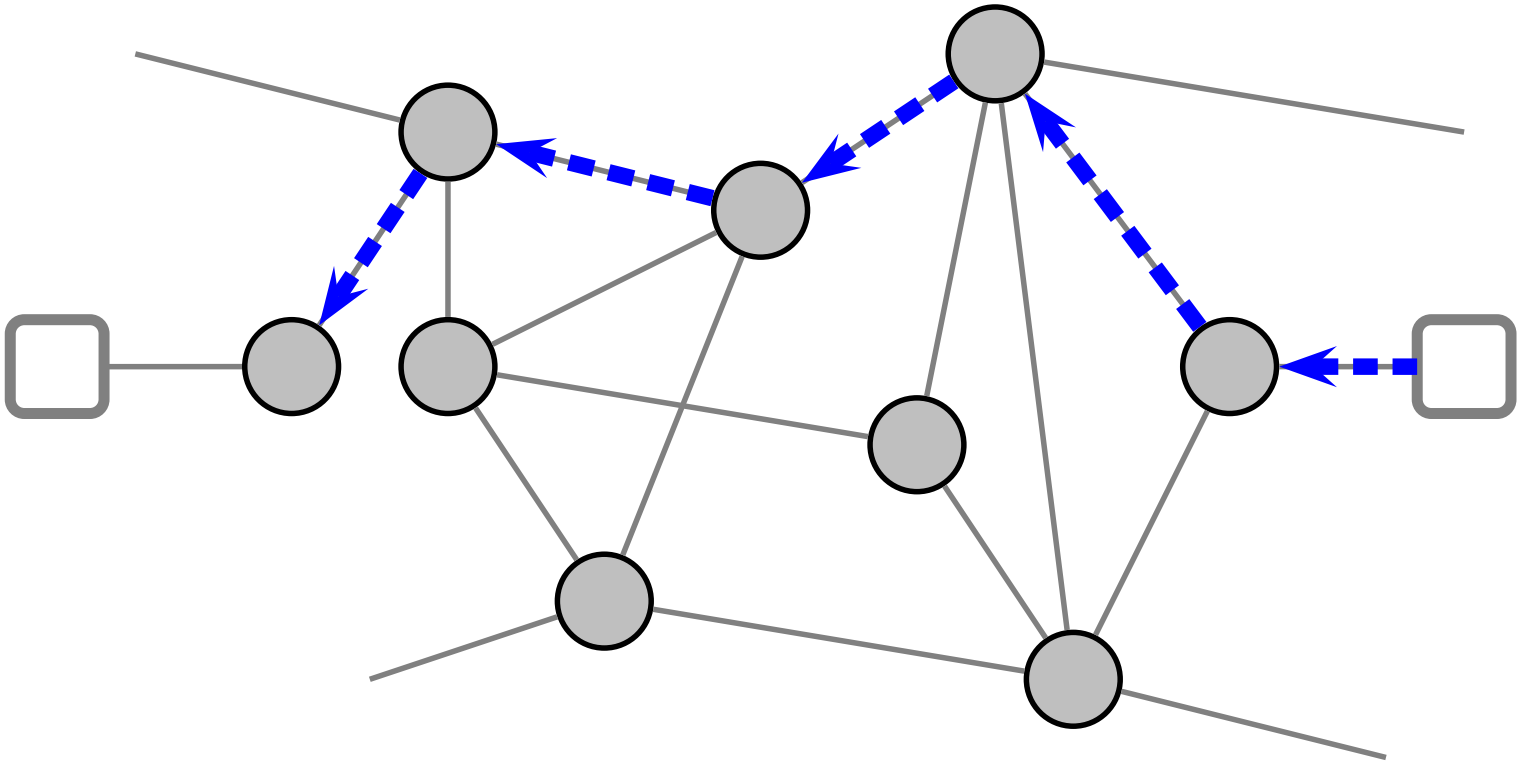
- Potentially *multiple paths* for the same source/destination
- Potentially *asymmetric paths*

Datagram Network



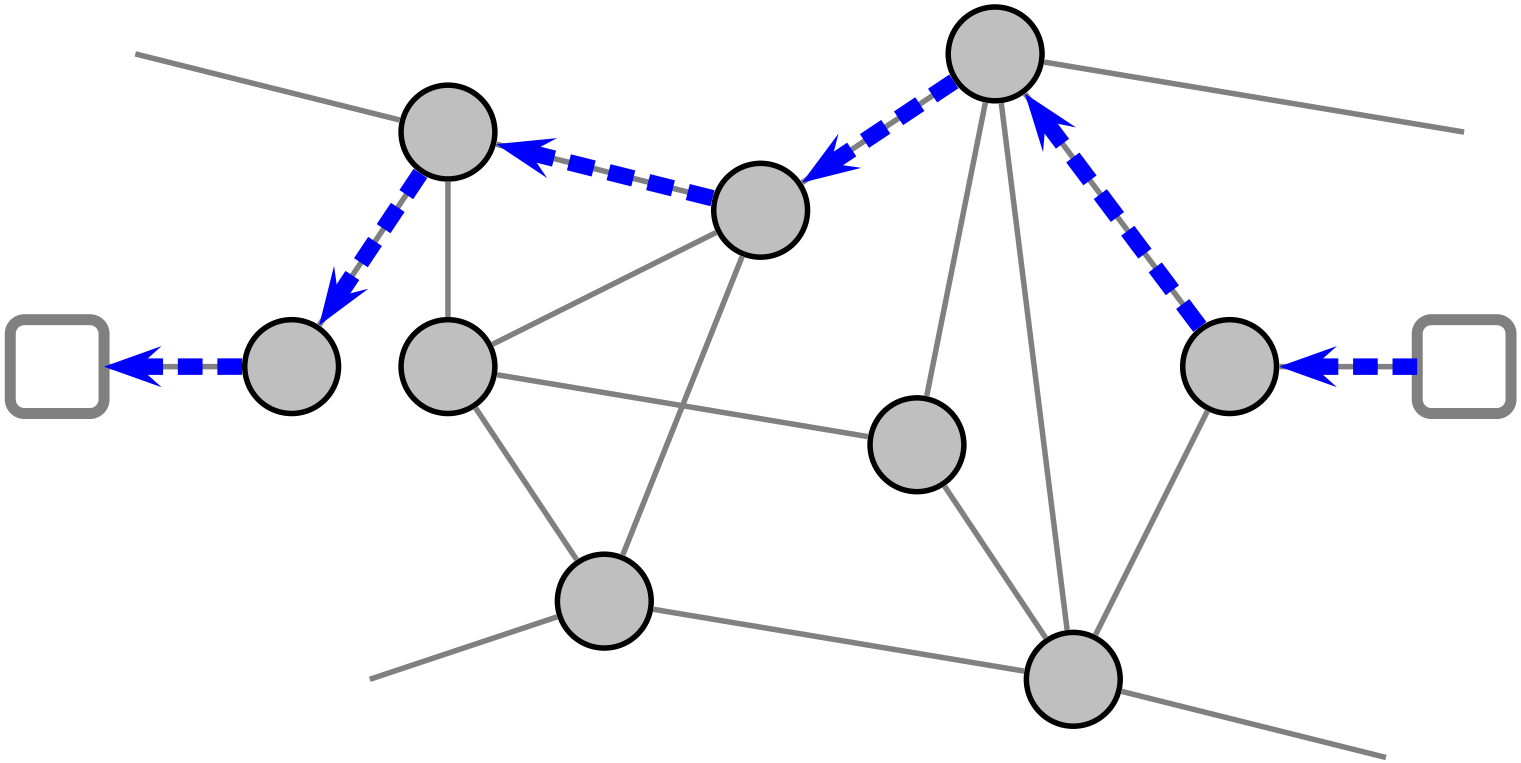
- Potentially *multiple paths* for the same source/destination
- Potentially *asymmetric paths*

Datagram Network



- Potentially *multiple paths* for the same source/destination
- Potentially *asymmetric paths*

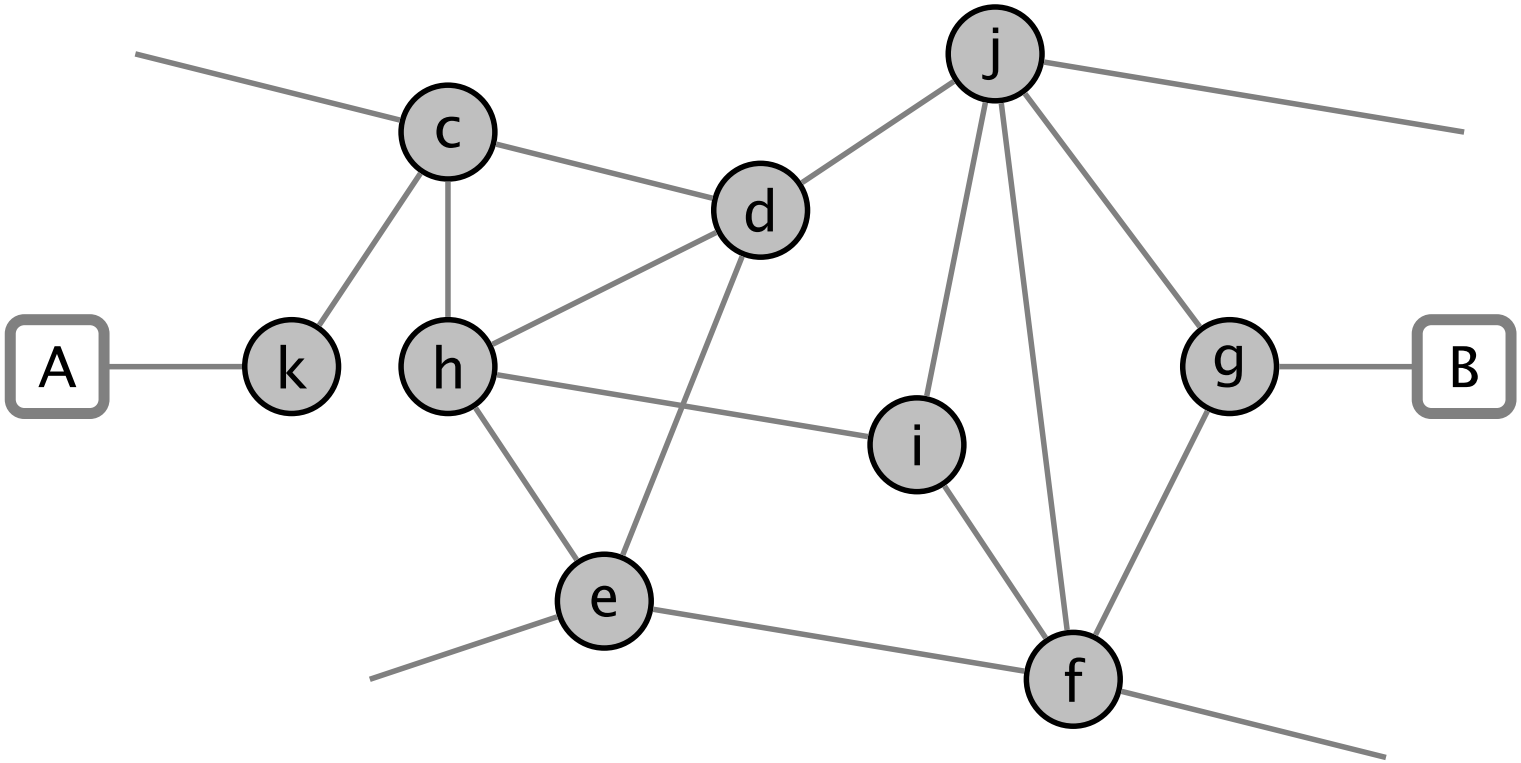
Datagram Network



- Potentially *multiple paths* for the same source/destination
- Potentially *asymmetric paths*

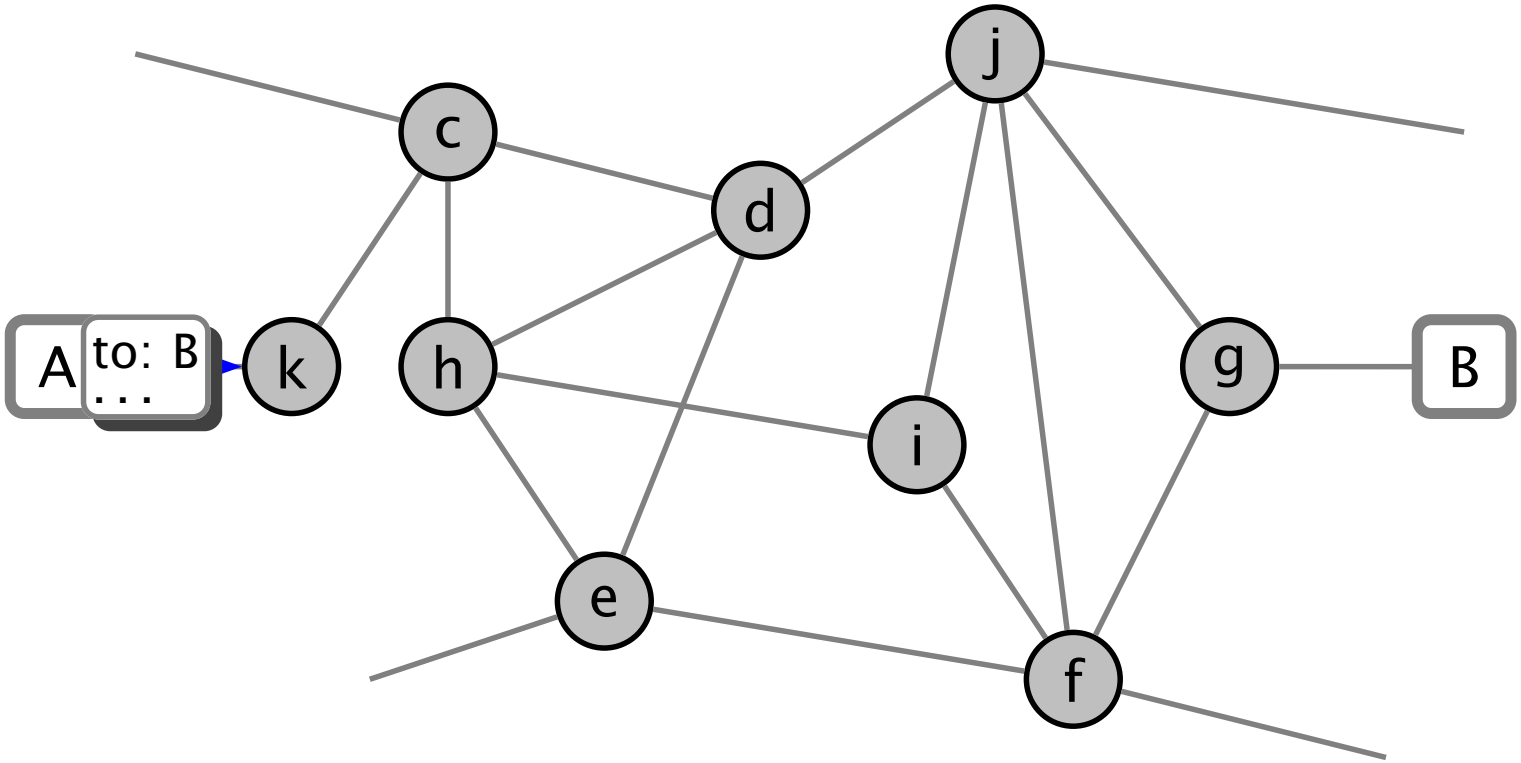
Forwarding

Forwarding



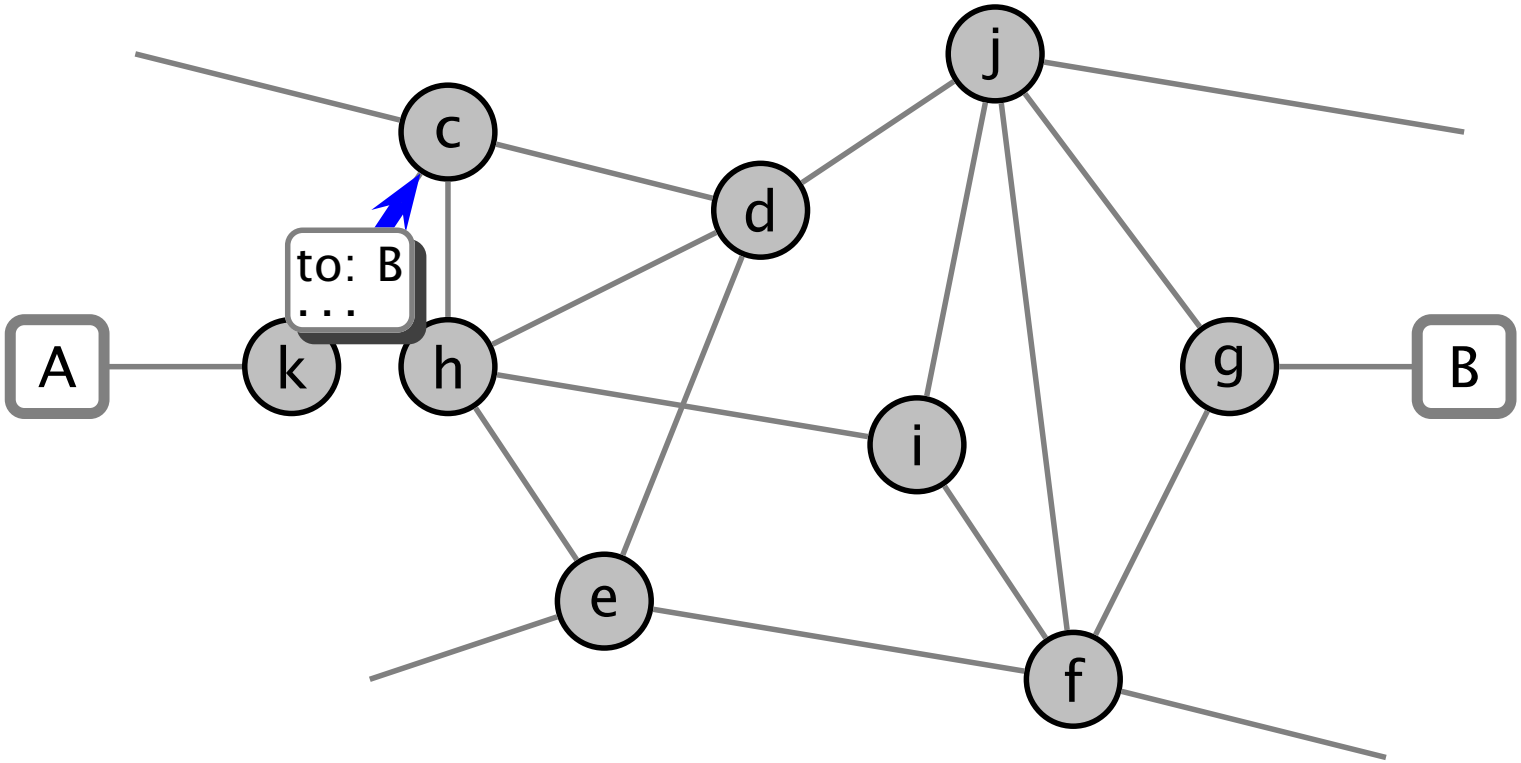
- *A* sends a datagram to *B*

Forwarding



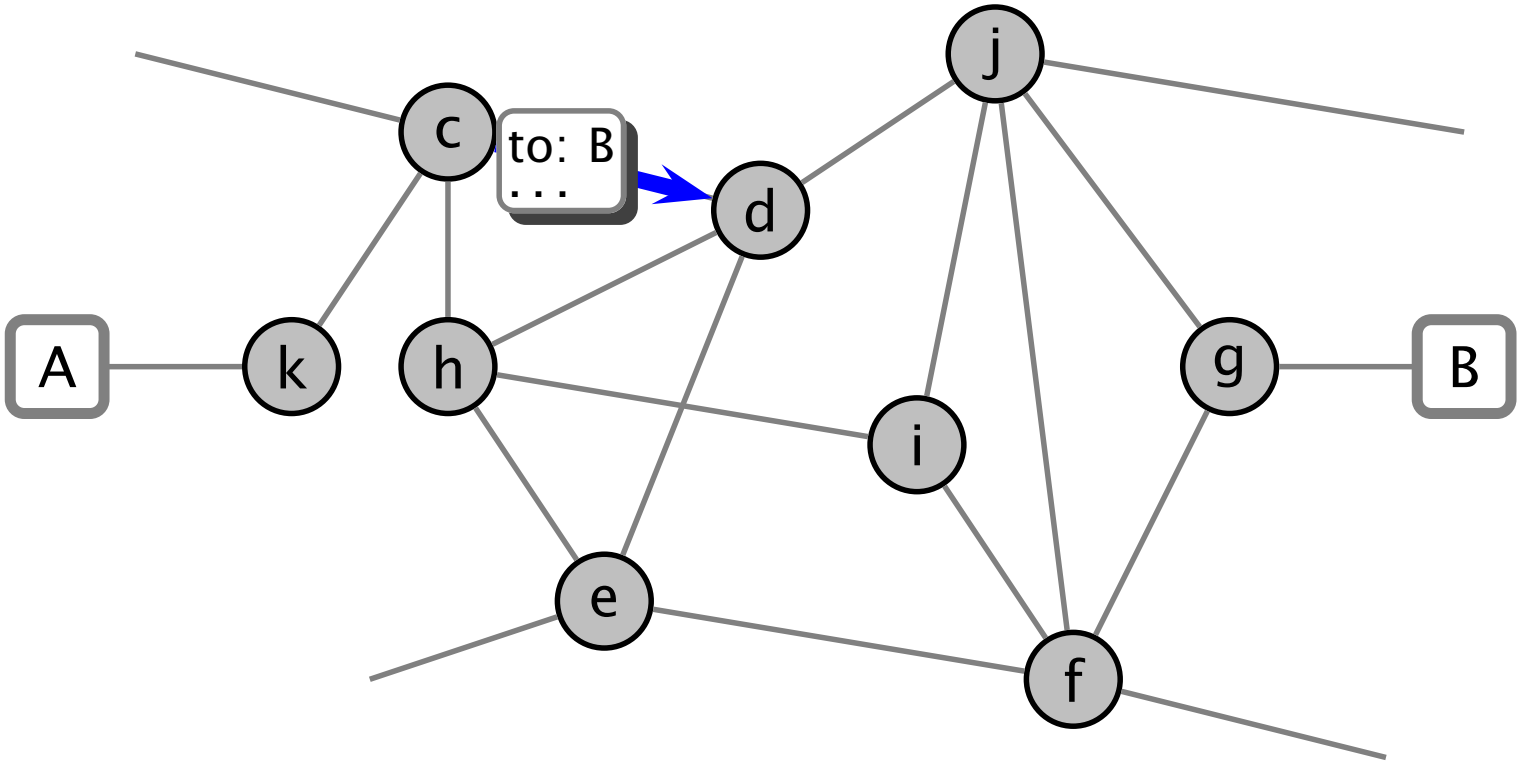
- *A* sends a datagram to *B*
- The datagram is *forwarded* towards *B*

Forwarding

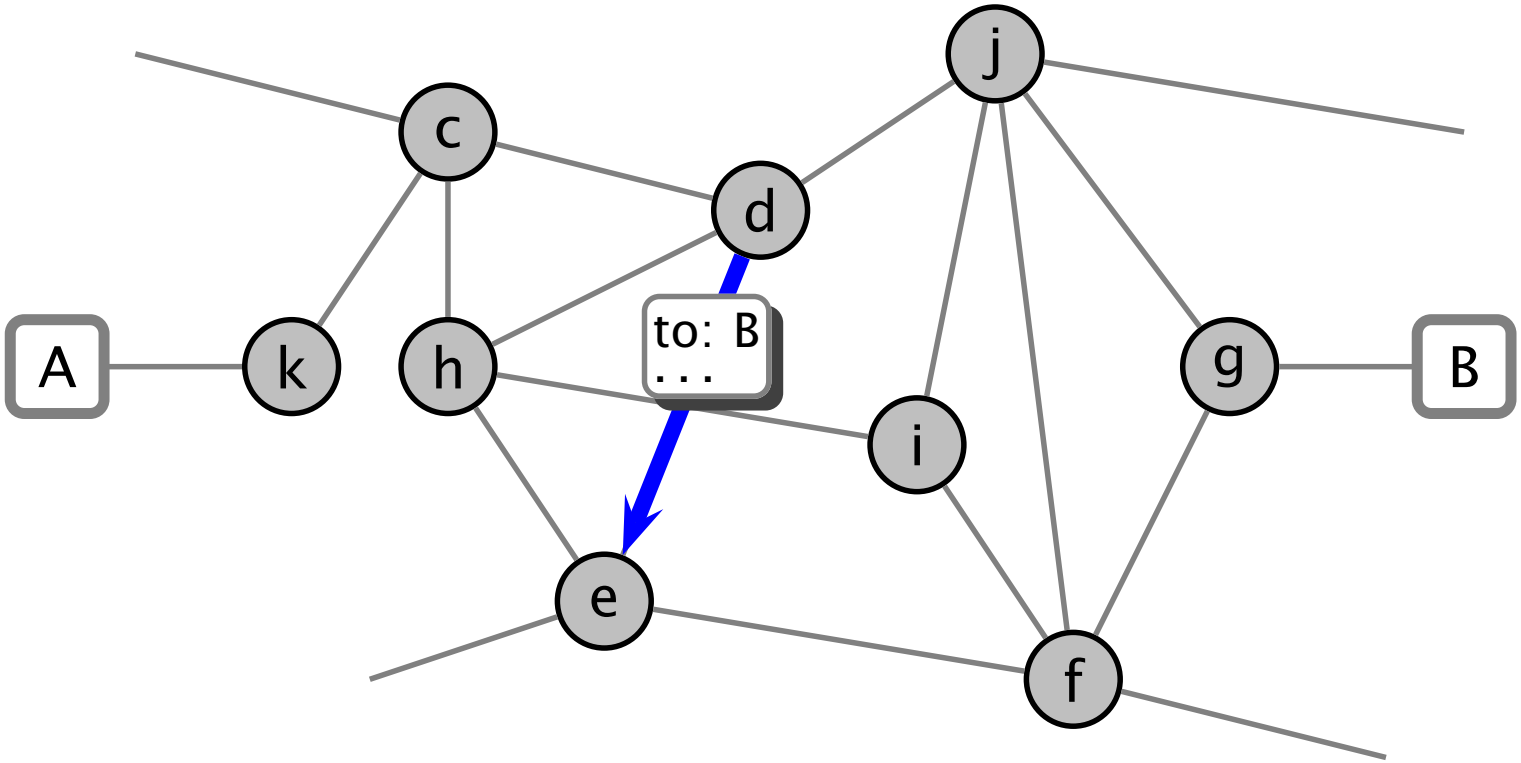


- *A* sends a datagram to *B*
- The datagram is *forwarded* towards *B*

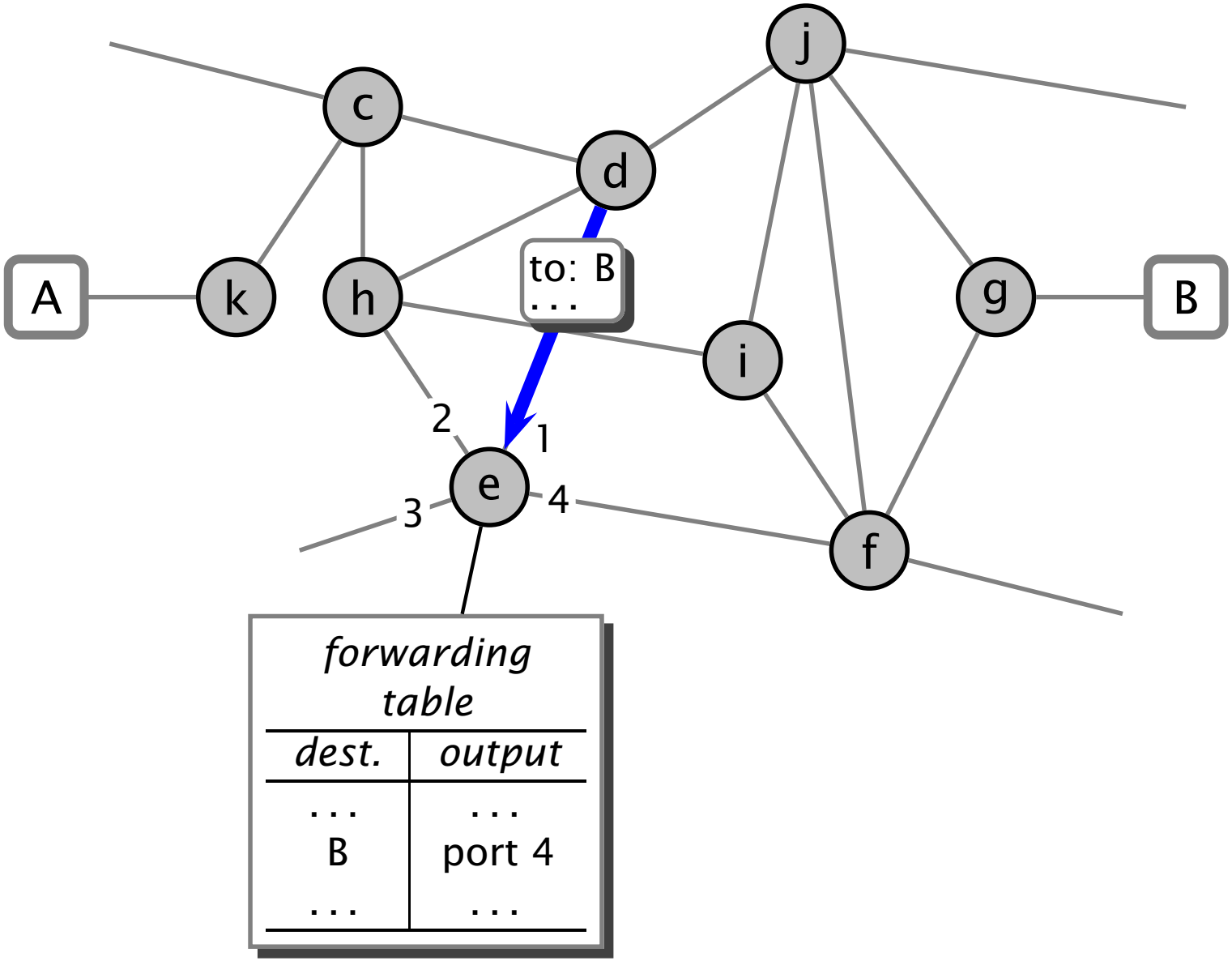
Forwarding



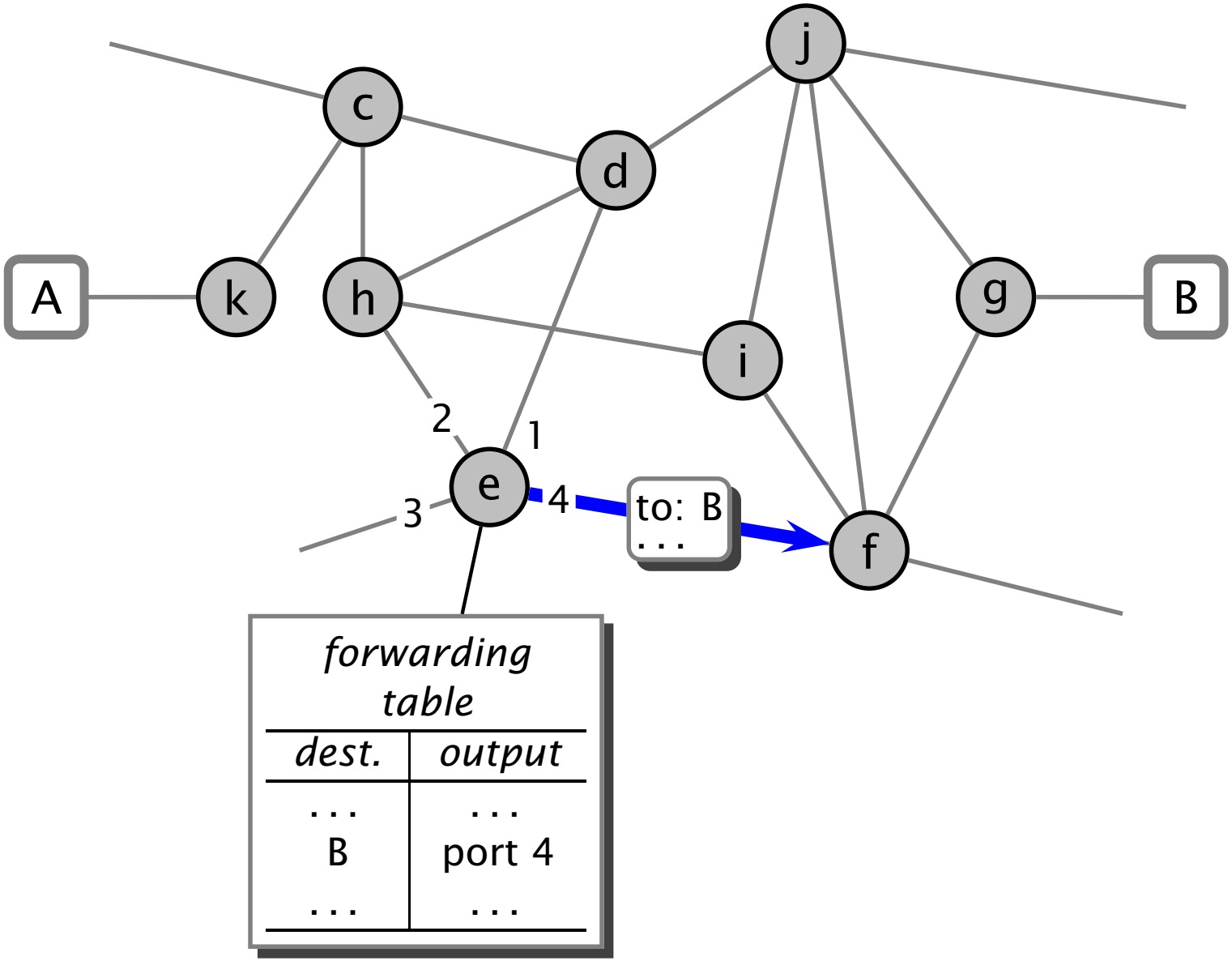
Forwarding



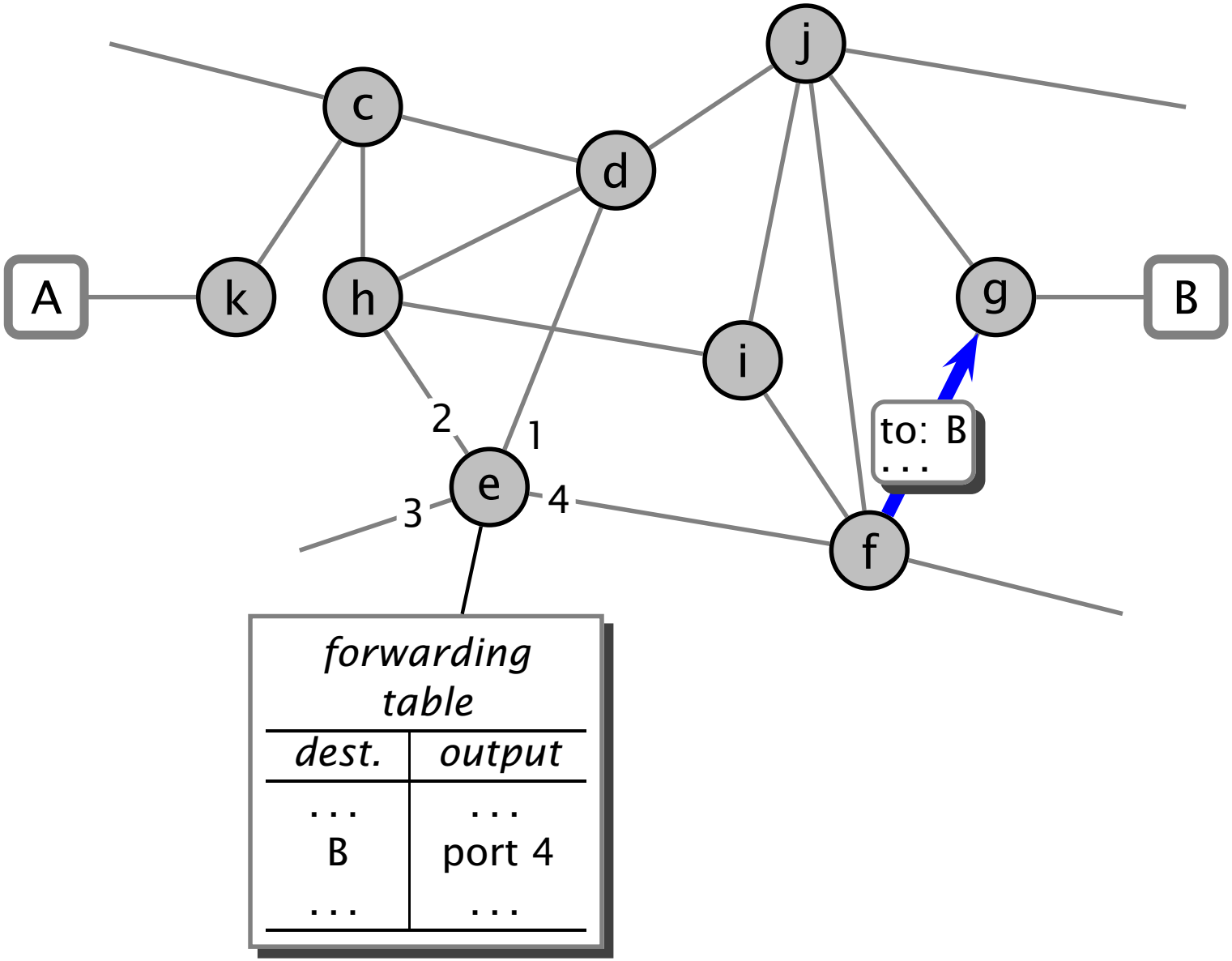
Forwarding



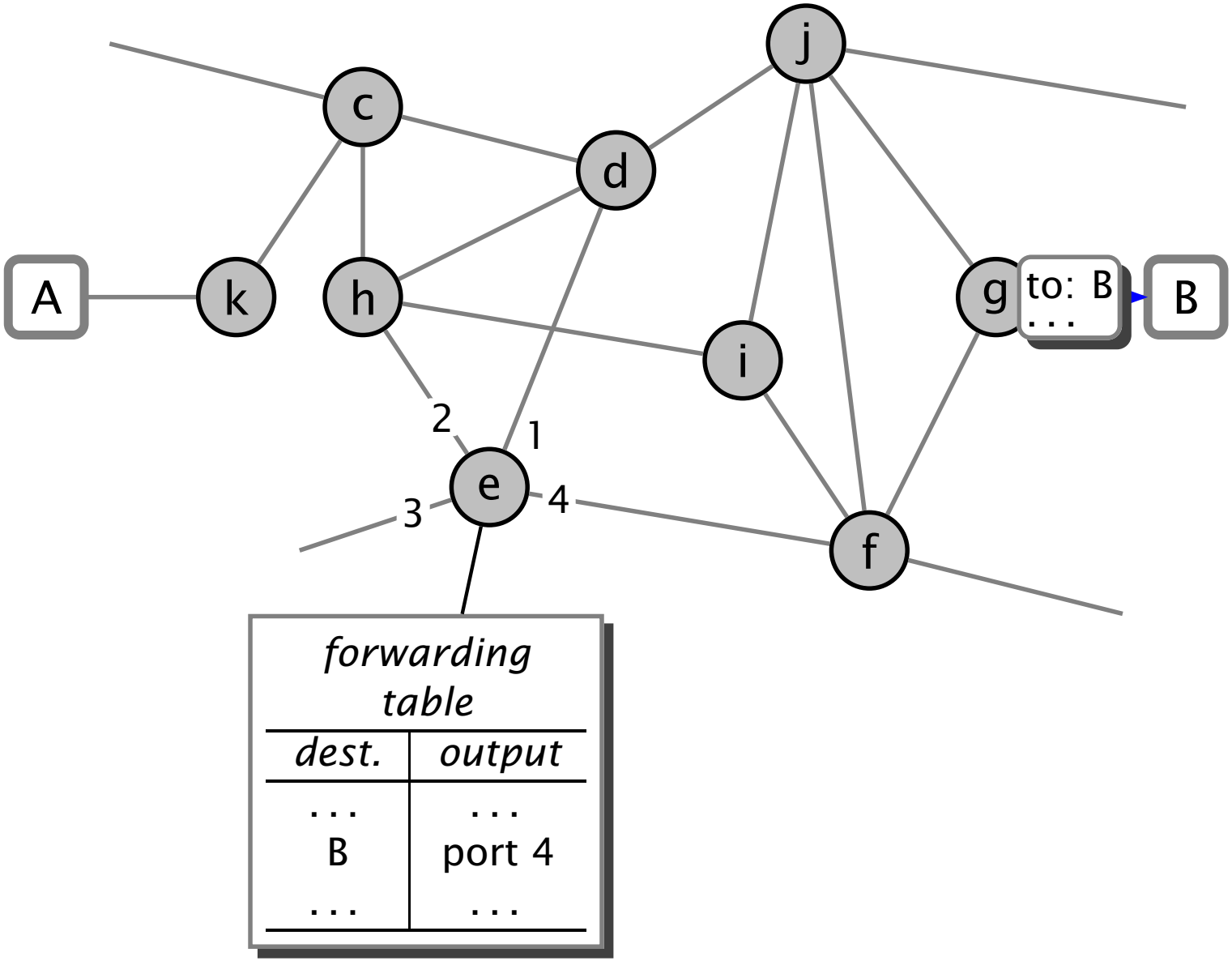
Forwarding



Forwarding



Forwarding



Forwarding

- *Input:* datagram destination

Forwarding

- *Input:* datagram destination
- *Output:* output port

Forwarding

- *Input:* datagram destination
- *Output:* output port
- Simple design: “forwarding table”

Forwarding

- *Input:* datagram destination
- *Output:* output port
- Simple design: “forwarding table”
- Issues

Forwarding

- *Input:* datagram destination
- *Output:* output port
- Simple design: “forwarding table”
- Issues
  - how big is the forwarding table?

Forwarding

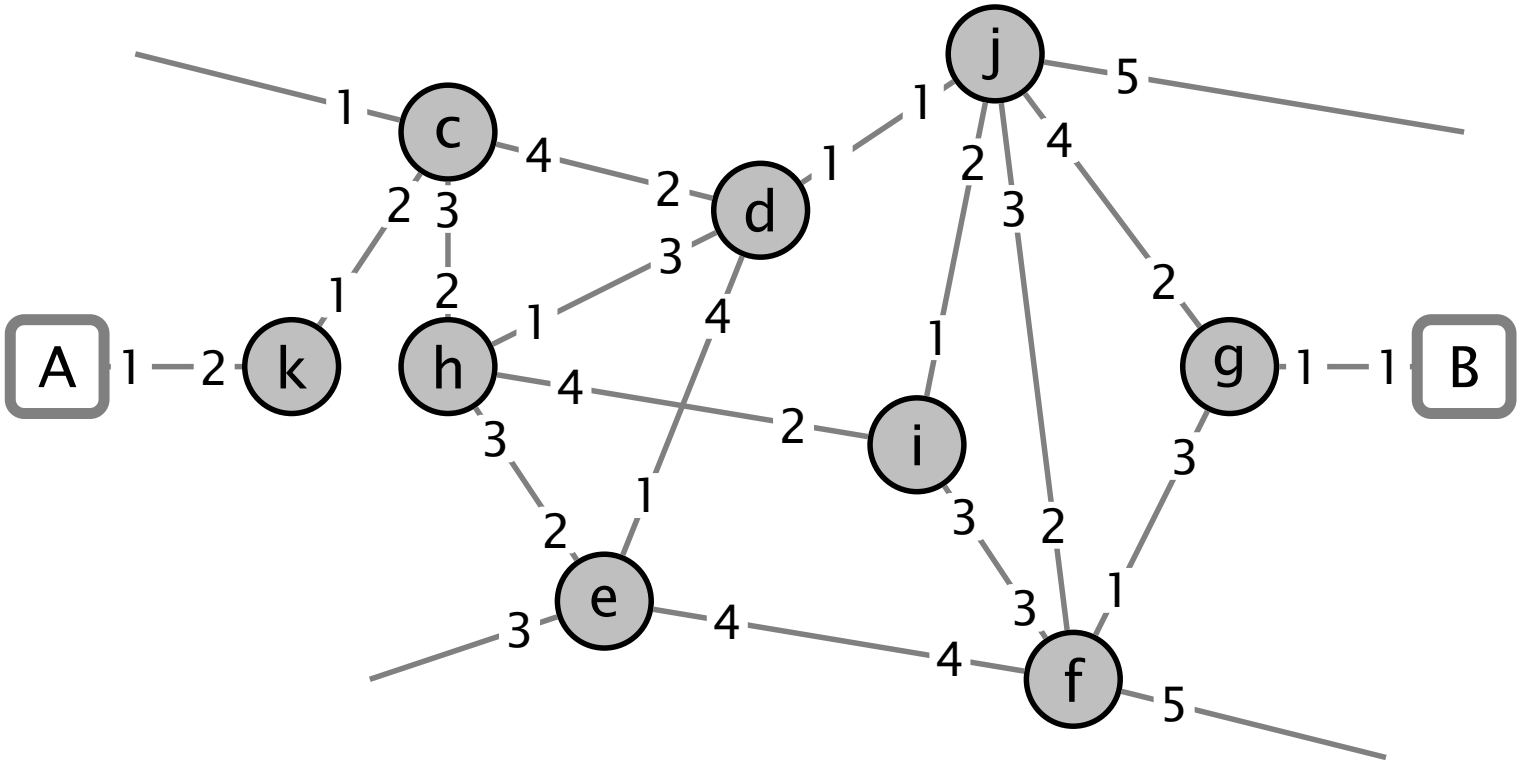
- *Input:* datagram destination
- *Output:* output port
- Simple design: “forwarding table”
- Issues
  - how big is the forwarding table?
  - how fast does the router have to forward datagrams?

Forwarding

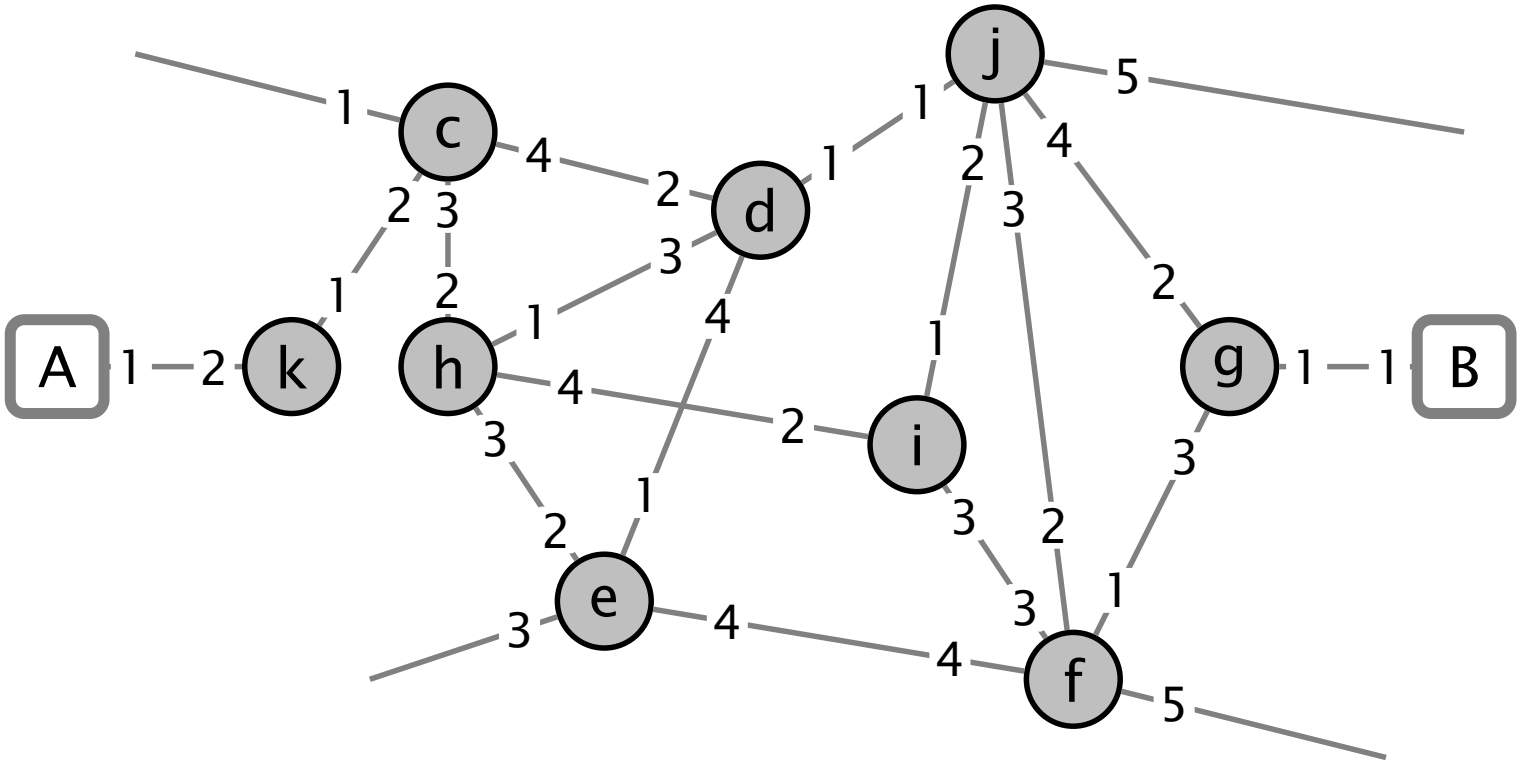
- *Input:* datagram destination
- *Output:* output port
- Simple design: “forwarding table”
- Issues
  - how big is the forwarding table?
  - how fast does the router have to forward datagrams?
  - how does the router build and maintain the forwarding table?

Routing

Routing



Routing

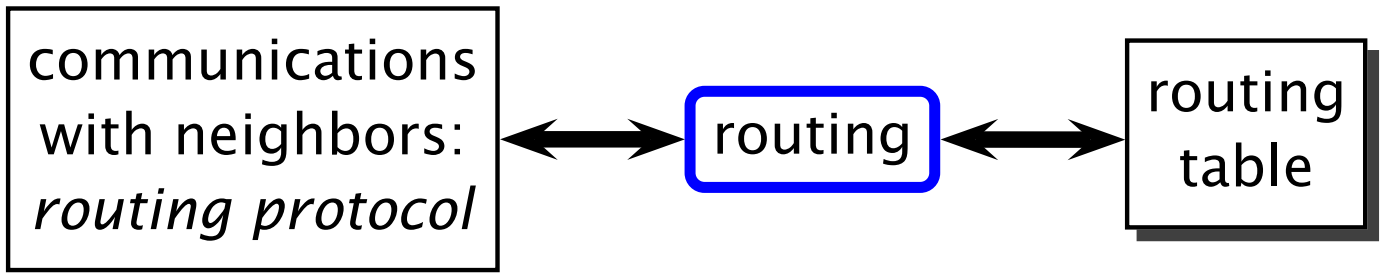


router k	
A	2
B	1

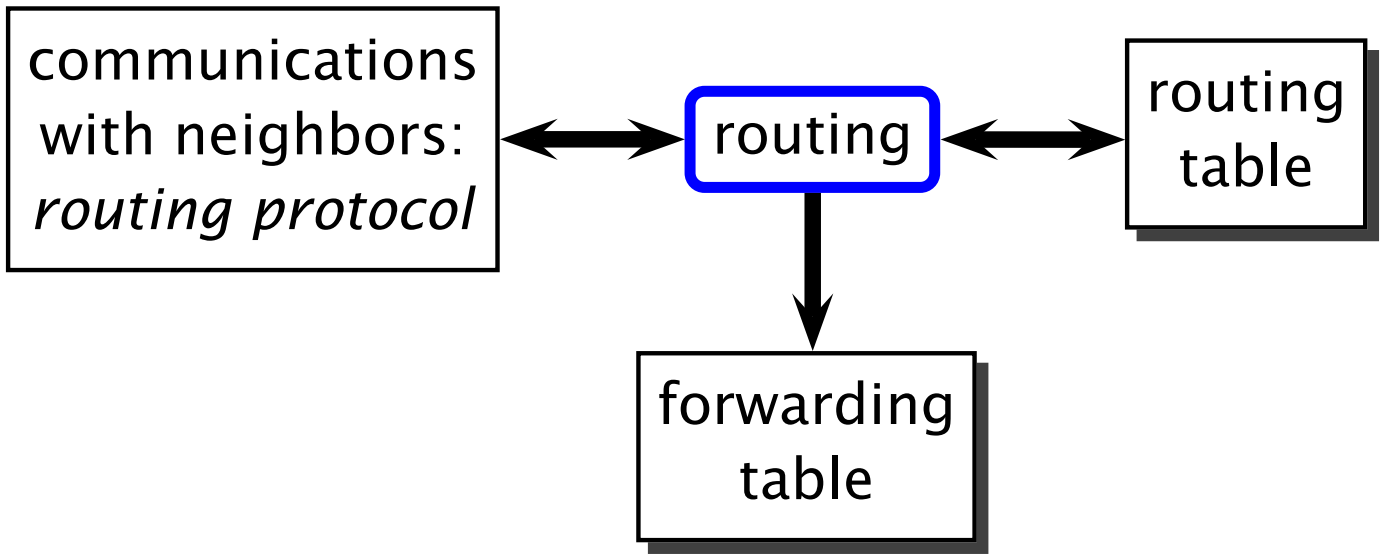
. . .

Router Functions

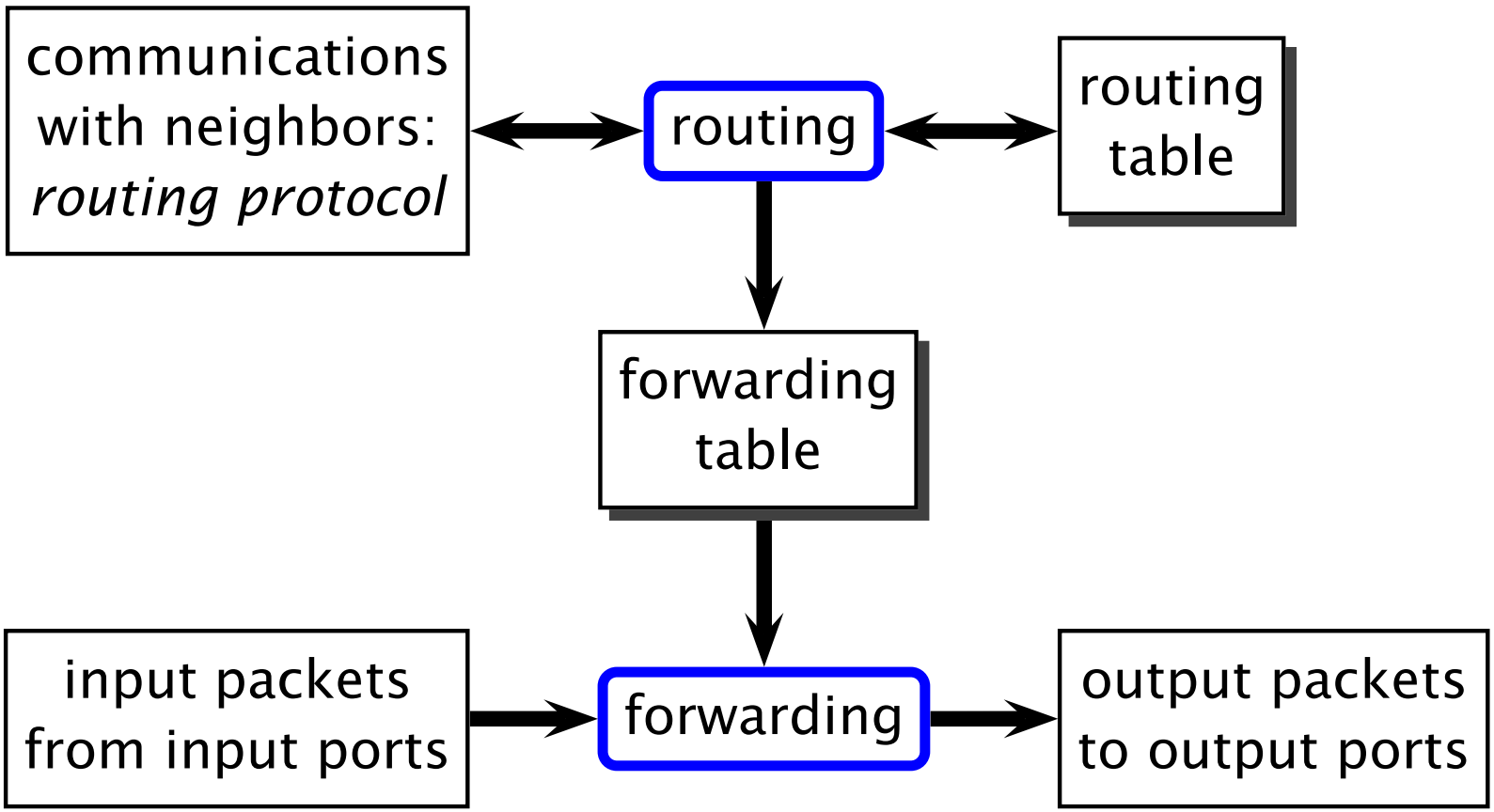
Router Functions



Router Functions

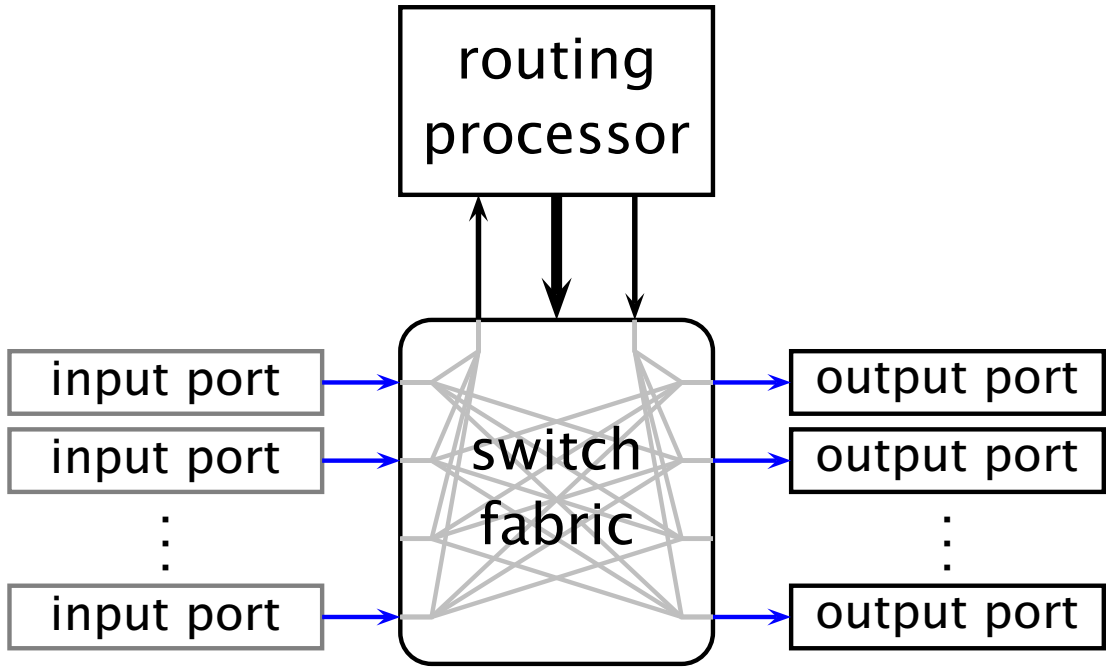


Router Functions

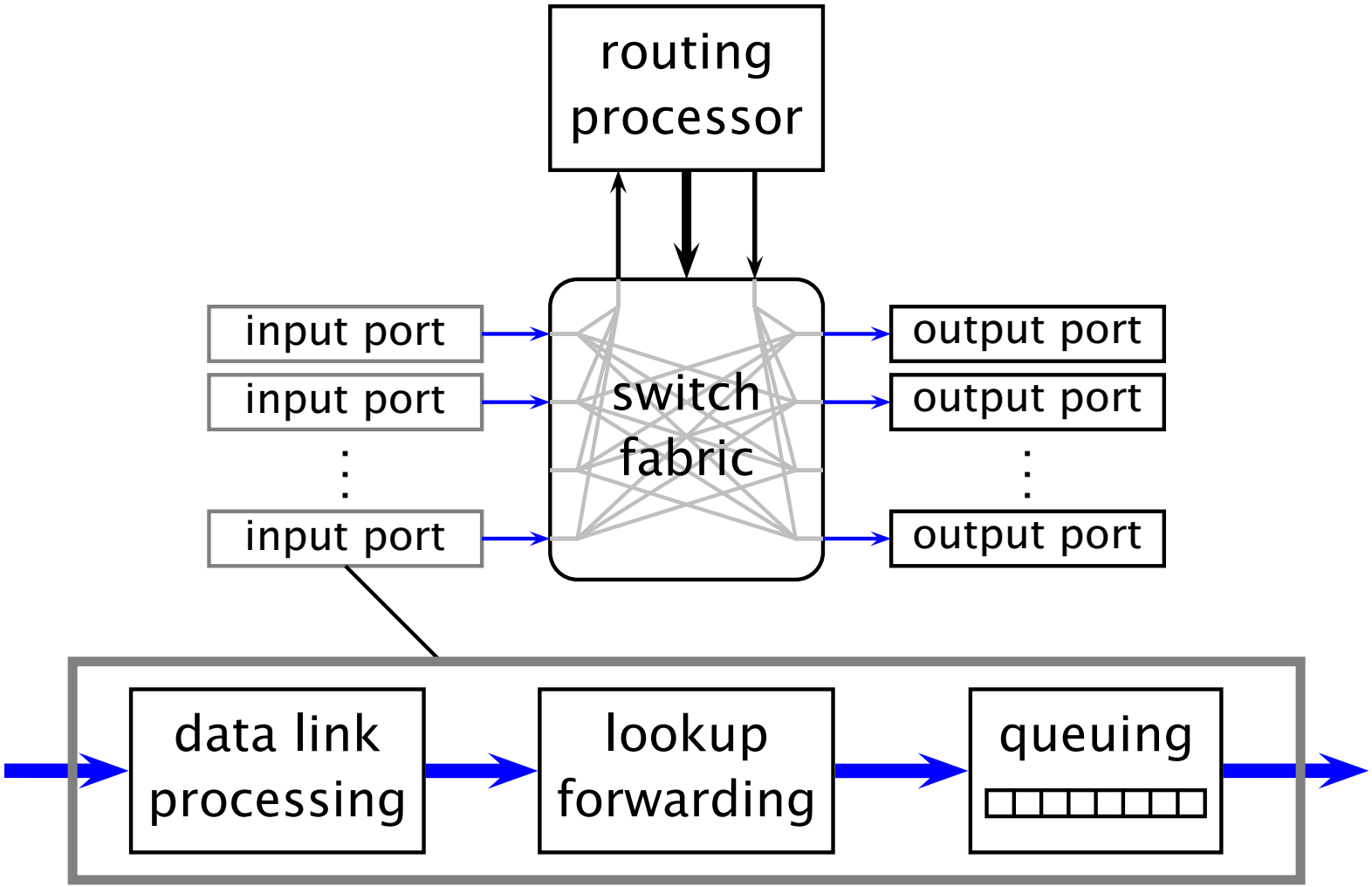


Anatomy of a Router

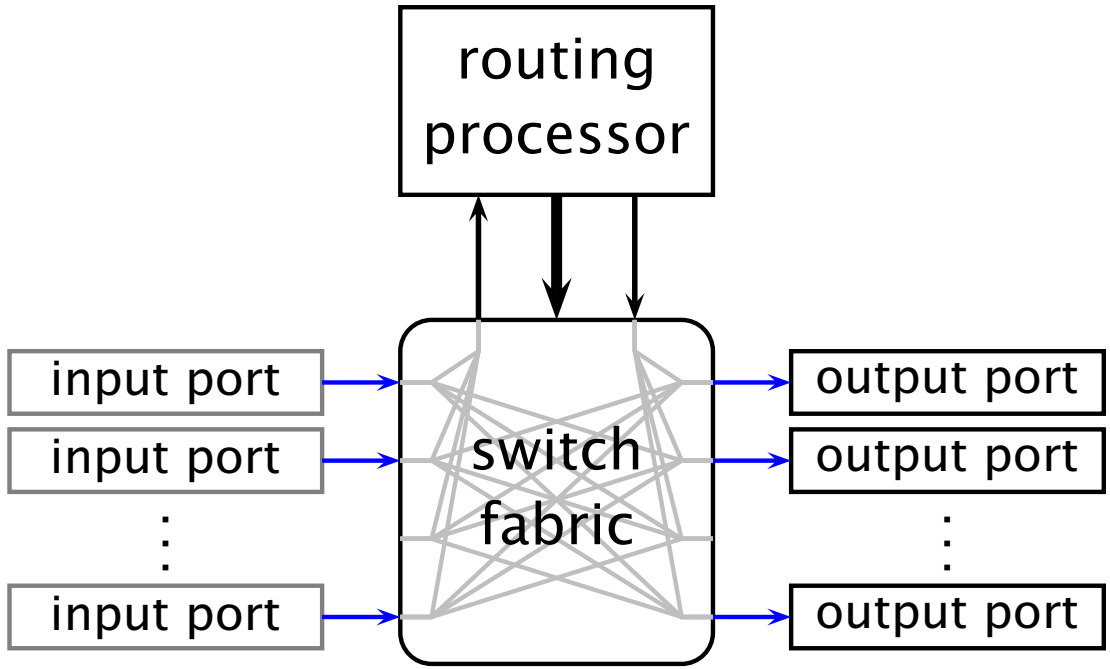
Anatomy of a Router



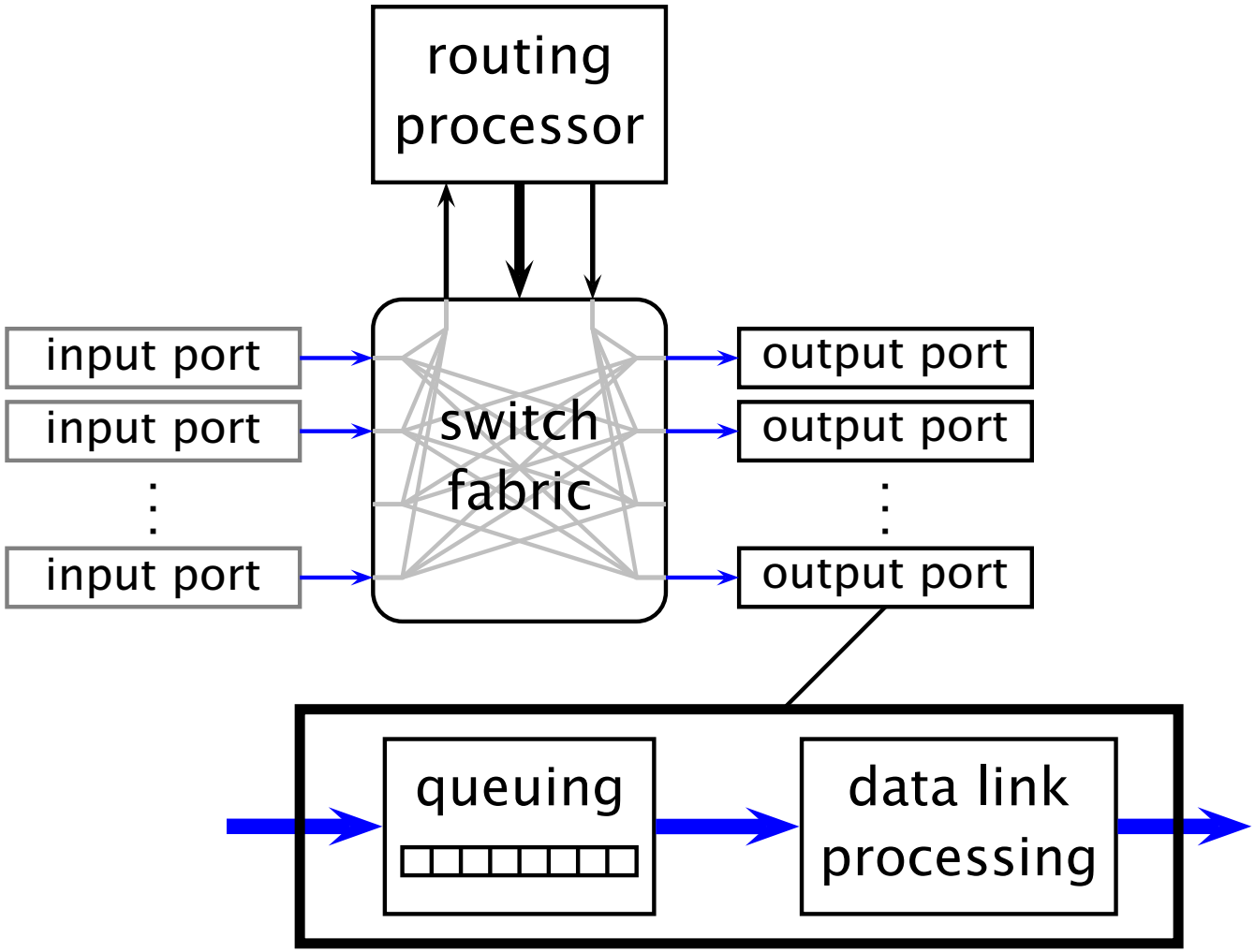
Anatomy of a Router



Anatomy of a Router



Anatomy of a Router



Queuing

- Where does queuing occur?

Queuing

- Where does queuing occur?
- Input ports
  - queuing may occur here if the switching fabric is slower than the aggregate speed of all the input lines. I.e., $R_S < nR_{in}$

Queuing

- Where does queuing occur?
- Input ports
  - queuing may occur here if the switching fabric is slower than the aggregate speed of all the input lines. I.e., $R_S < nR_{in}$
- Output ports
  - queuing may occur here because of the limited throughput of the output link. I.e., $R_{out} < \min(R_S, nR_{in})$

Queuing

- What happens when packets queue up in a router?

Queuing

- What happens when packets queue up in a router?
- *Scheduling:* deciding which packets to process

Queuing

- What happens when packets queue up in a router?
- *Scheduling:* deciding which packets to process
  - *first-come-first-served*

Queuing

- What happens when packets queue up in a router?
- *Scheduling:* deciding which packets to process
  - *first-come-first-served*
  - *weighted fair queuing:* the router tries to be balance traffic evenly among the different end-to-end connections. Essential to implement *quality-of-service guarantees*

Queuing

- What happens when packets queue up in a router?
- *Scheduling:* deciding which packets to process
  - *first-come-first-served*
  - *weighted fair queuing:* the router tries to be balance traffic evenly among the different end-to-end connections. Essential to implement *quality-of-service guarantees*
- Deciding when to drop packets, and which packets to drop

Queuing

- What happens when packets queue up in a router?
- *Scheduling:* deciding which packets to process
  - *first-come-first-served*
  - *weighted fair queuing:* the router tries to be balance traffic evenly among the different end-to-end connections. Essential to implement *quality-of-service guarantees*
- Deciding when to drop packets, and which packets to drop
  - *drop tail:* drop arriving packets when queues are full

Queuing

- What happens when packets queue up in a router?
- *Scheduling:* deciding which packets to process
  - *first-come-first-served*
  - *weighted fair queuing:* the router tries to be balance traffic evenly among the different end-to-end connections. Essential to implement *quality-of-service guarantees*
- Deciding when to drop packets, and which packets to drop
  - *drop tail:* drop arriving packets when queues are full
  - *active queue management:* a set of policies and algorithms to decide when and how to drop or mark packets in the attempt to prevent congestion

Internet Network Layer

Internet Network Layer

- *Routing:* defining paths and compiling forwarding tables

Internet Network Layer

- *Routing:* defining paths and compiling forwarding tables
  - RIP
  - OSPF
  - BGP

Internet Network Layer

- *Routing:* defining paths and compiling forwarding tables
  - RIP
  - OSPF
  - BGP
- IP

Internet Network Layer

- *Routing:* defining paths and compiling forwarding tables
  - RIP
  - OSPF
  - BGP
- IP
  - addressing
  - datagram format
  - fragmentation and packet handling

Internet Network Layer

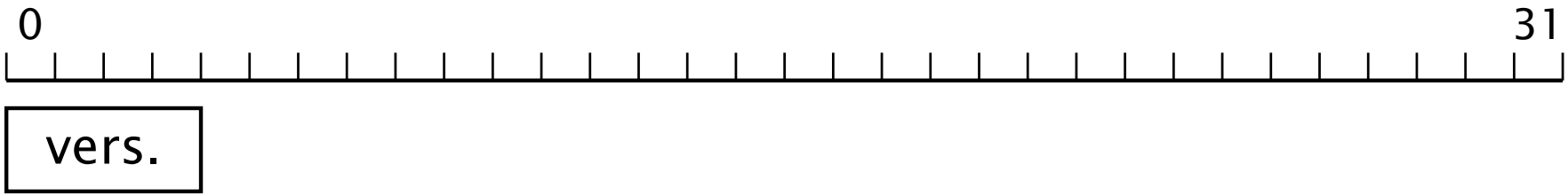
- *Routing:* defining paths and compiling forwarding tables
  - RIP
  - OSPF
  - BGP
- IP
  - addressing
  - datagram format
  - fragmentation and packet handling
- ICMP

Internet Network Layer

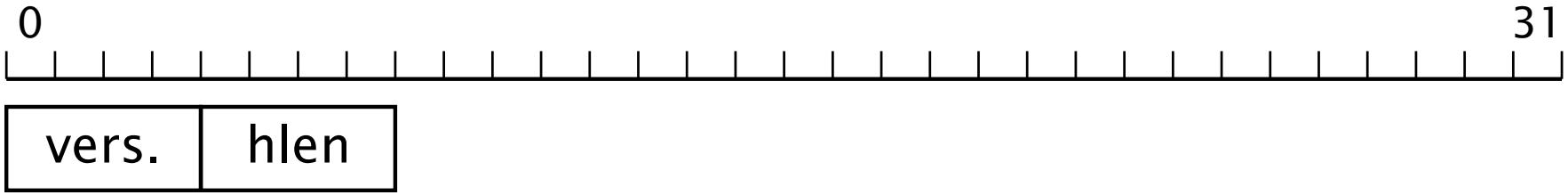
- *Routing:* defining paths and compiling forwarding tables
  - RIP
  - OSPF
  - BGP
- IP
  - addressing
  - datagram format
  - fragmentation and packet handling
- ICMP
  - error reporting
  - signaling

IPv4 Datagram Format

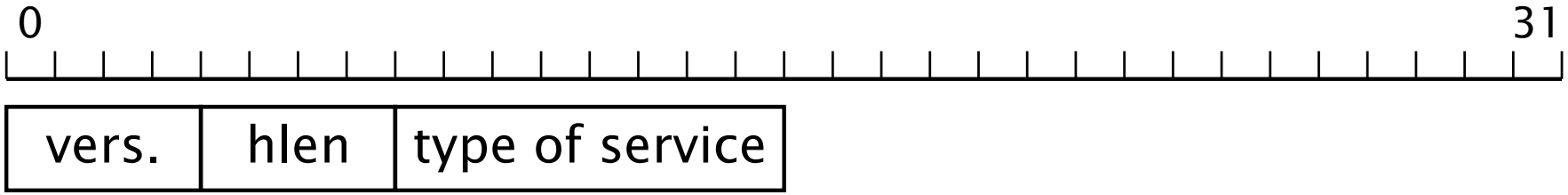
IPv4 Datagram Format



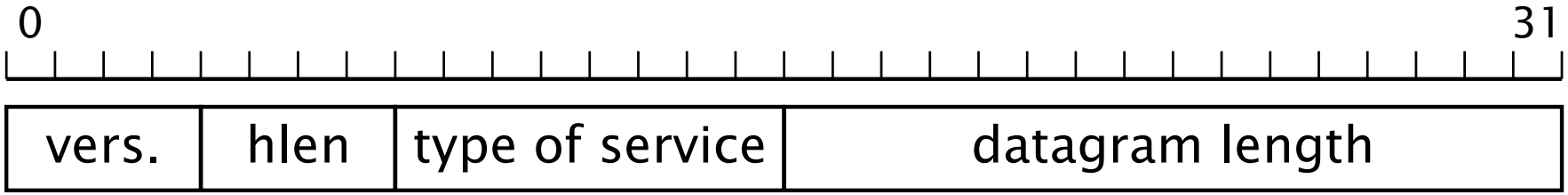
IPv4 Datagram Format



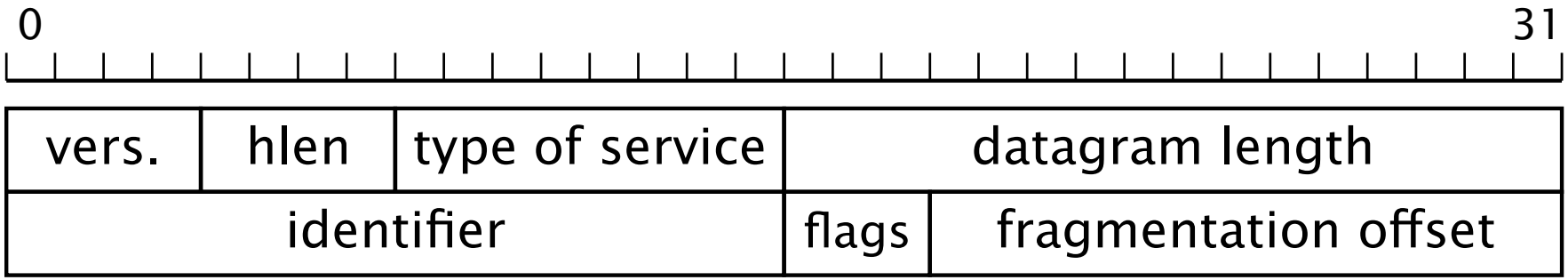
IPv4 Datagram Format



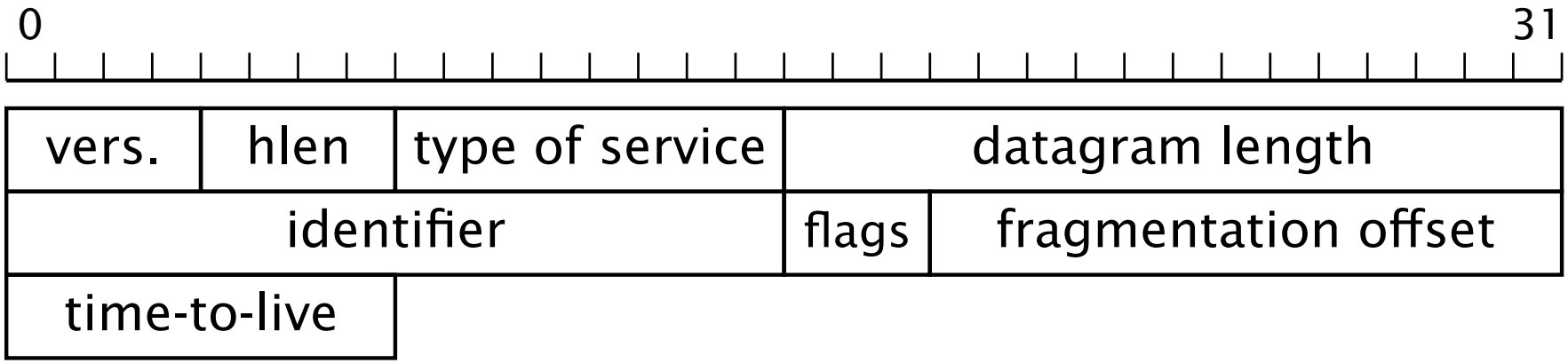
IPv4 Datagram Format



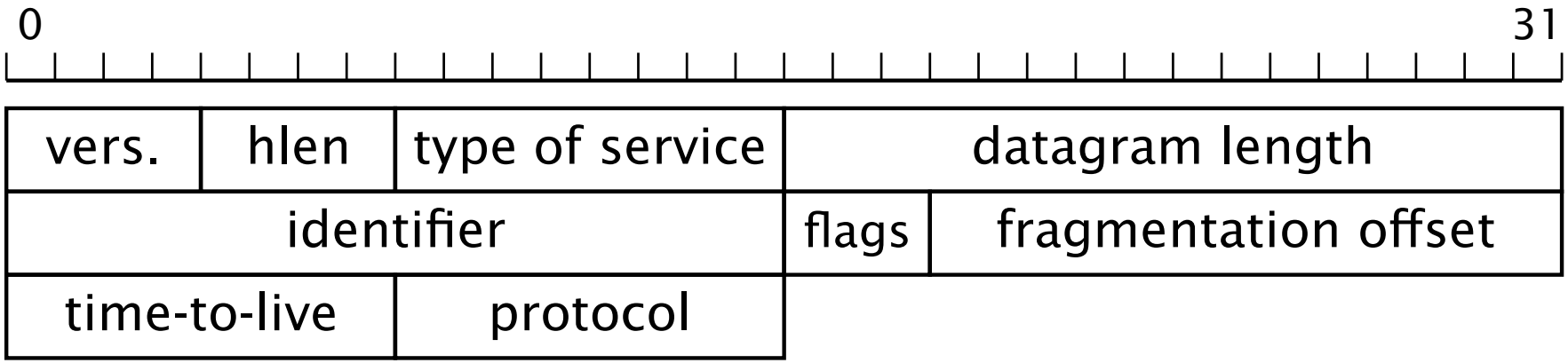
IPv4 Datagram Format



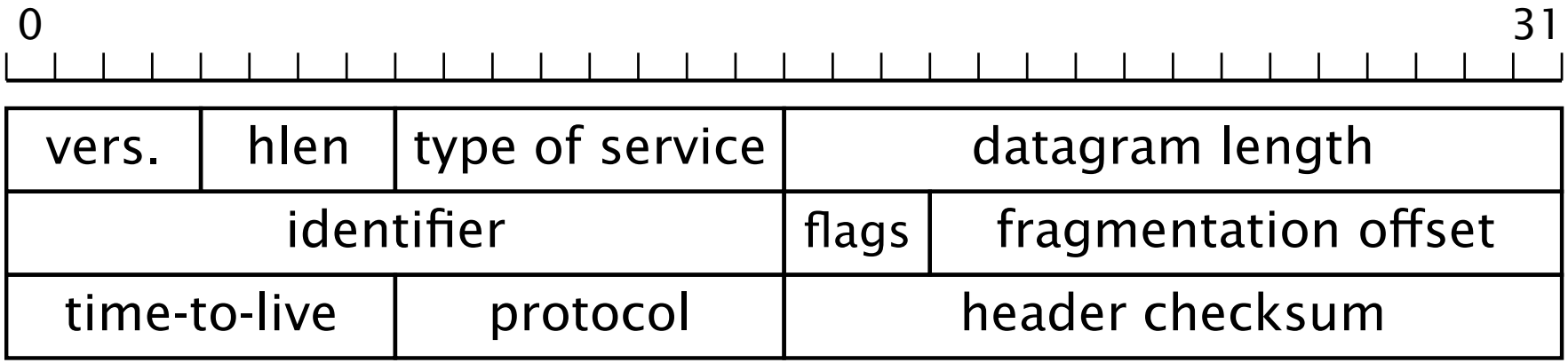
IPv4 Datagram Format



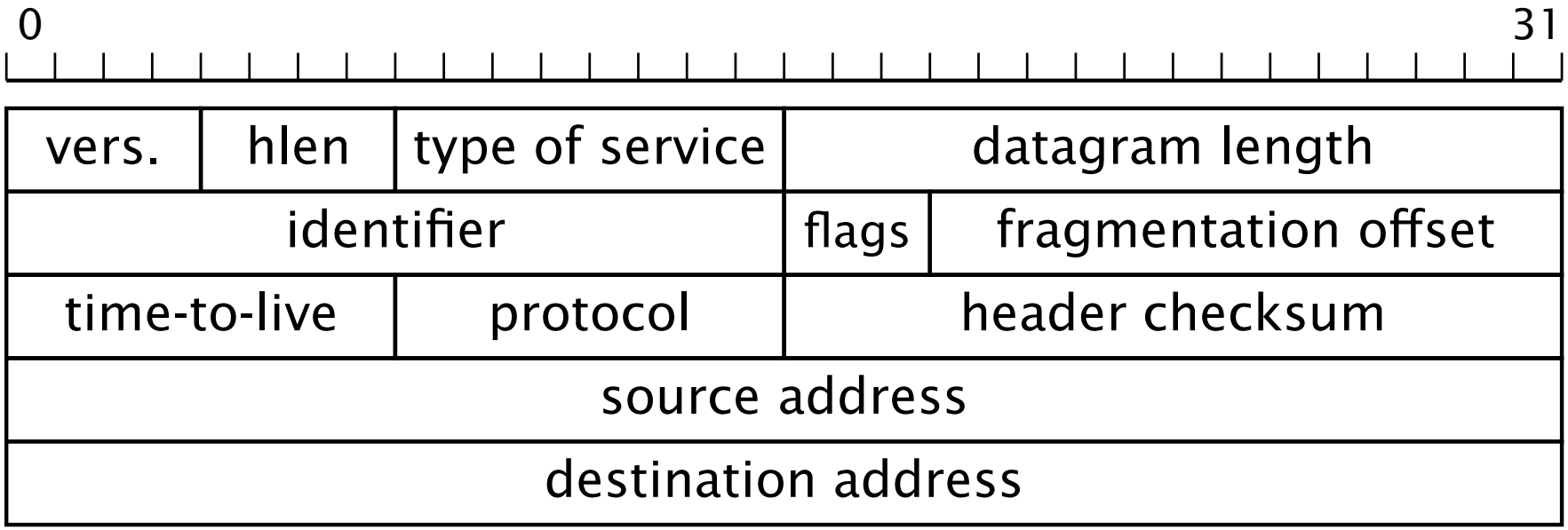
IPv4 Datagram Format



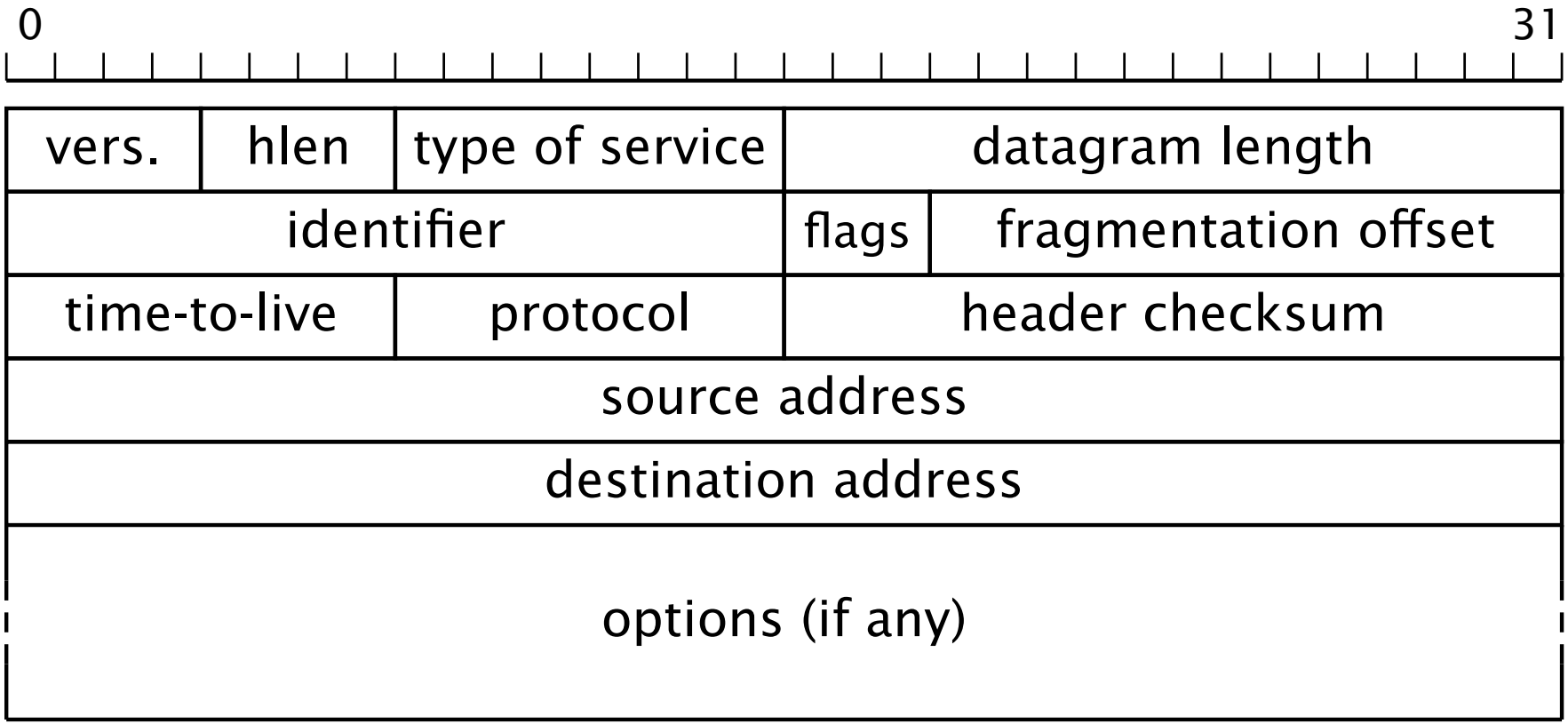
IPv4 Datagram Format



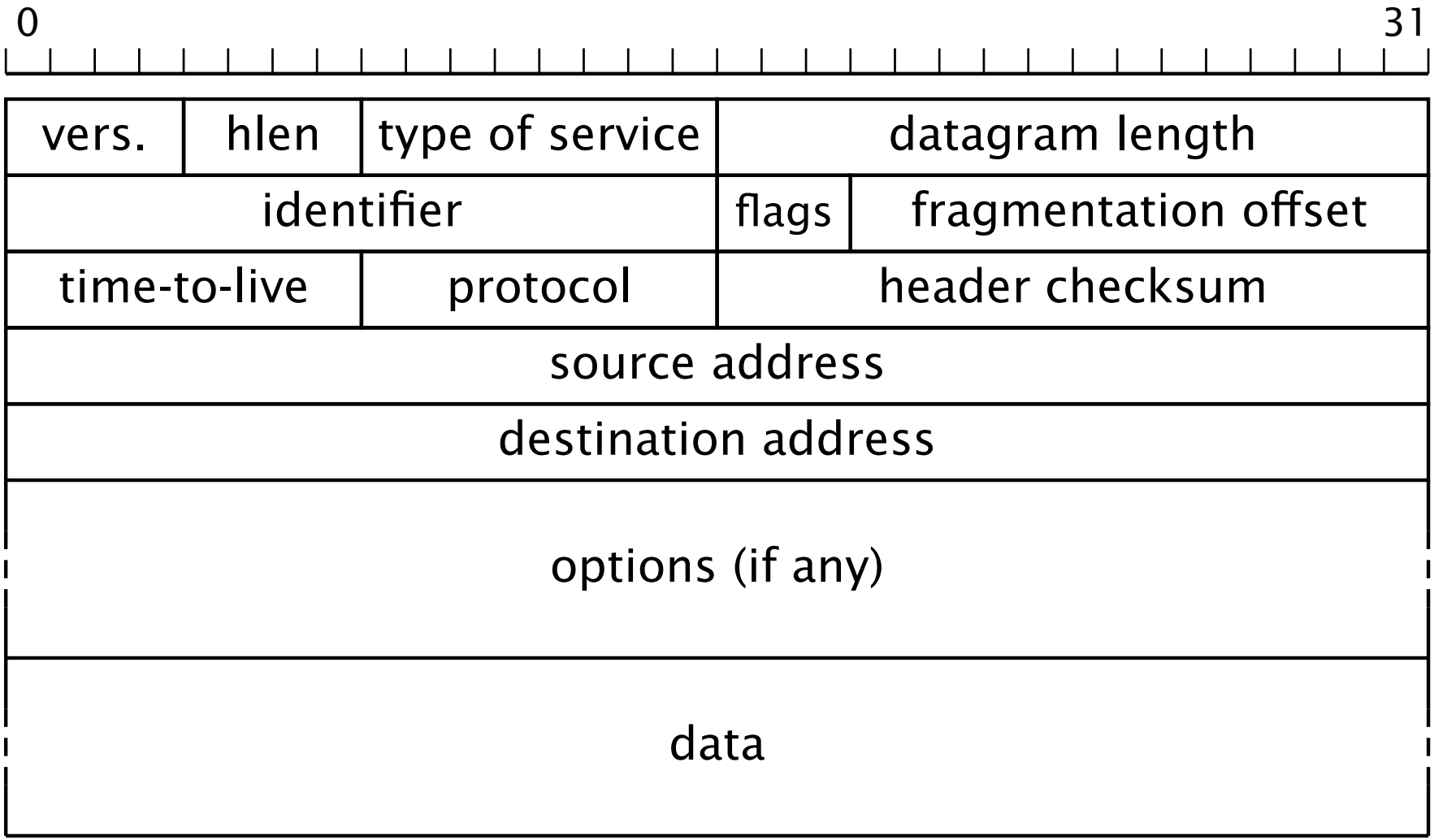
IPv4 Datagram Format



IPv4 Datagram Format

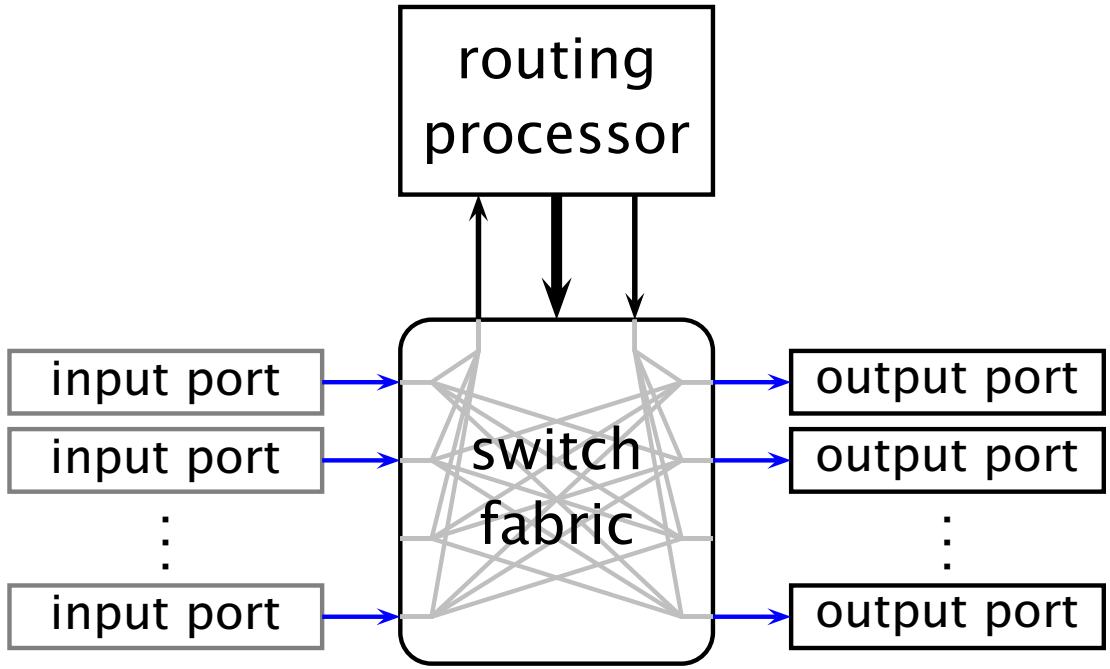


IPv4 Datagram Format

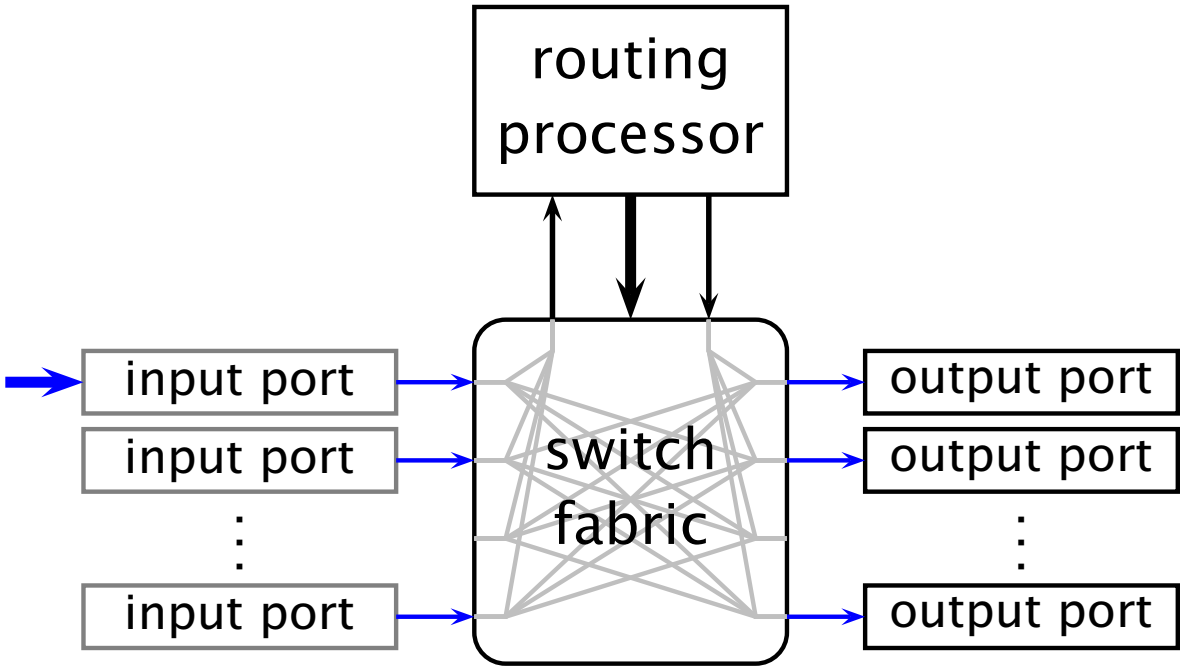


Fragmentation

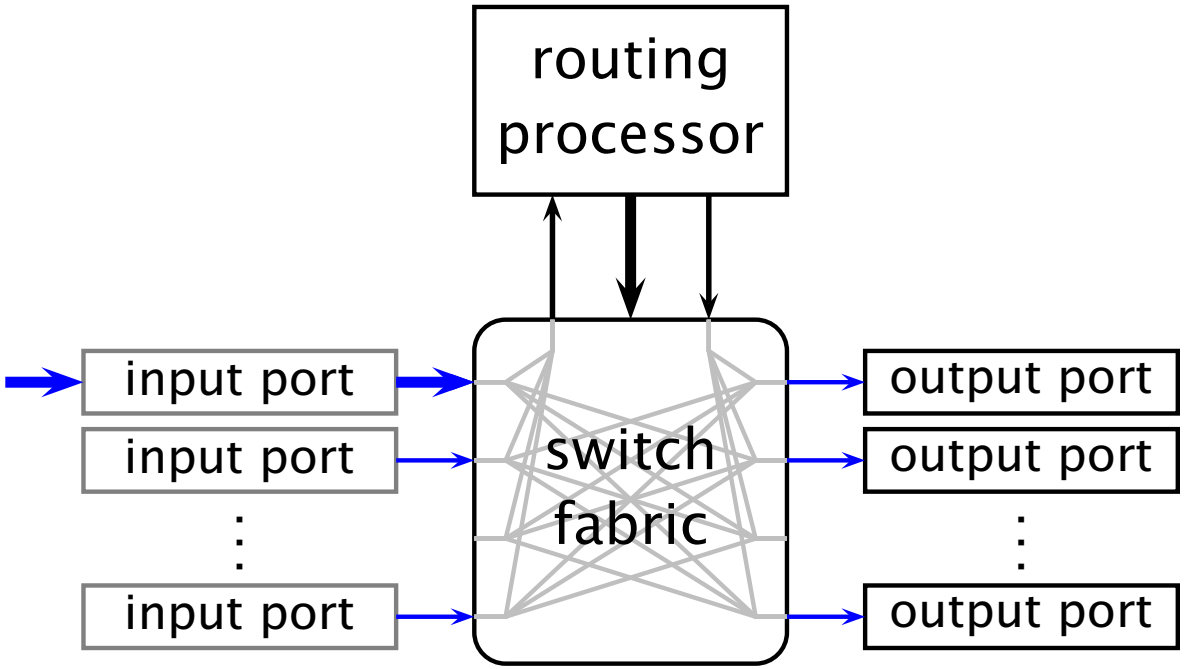
Fragmentation



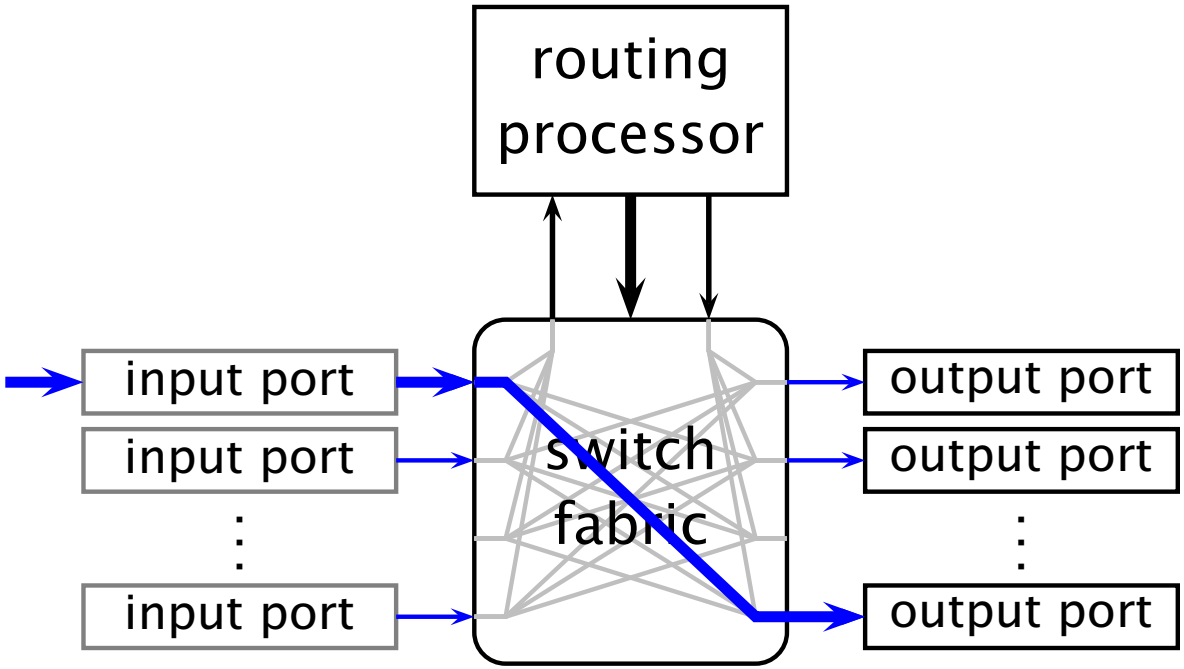
Fragmentation



Fragmentation

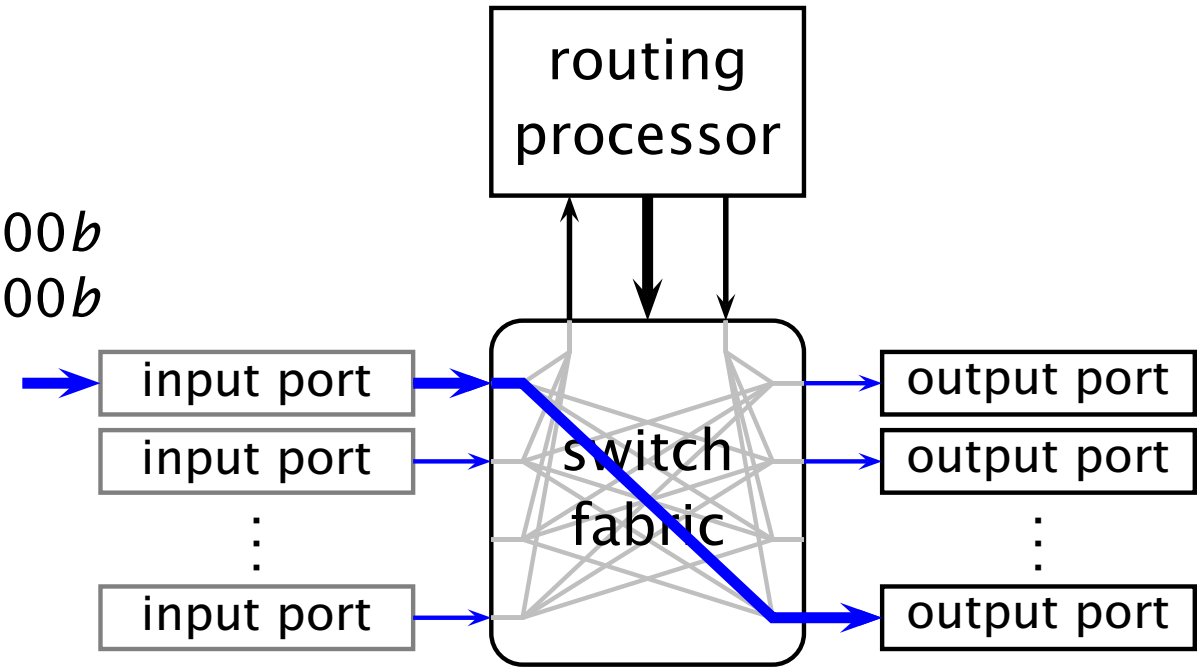


Fragmentation

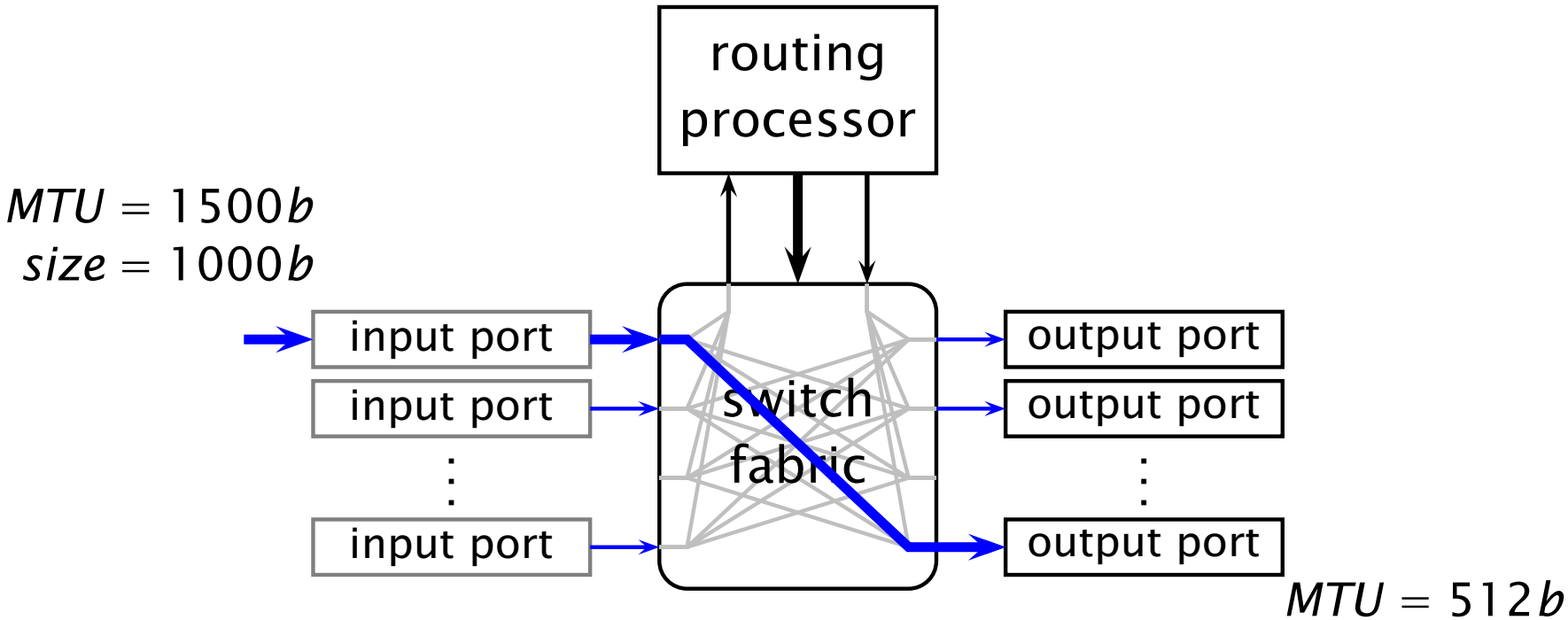


Fragmentation

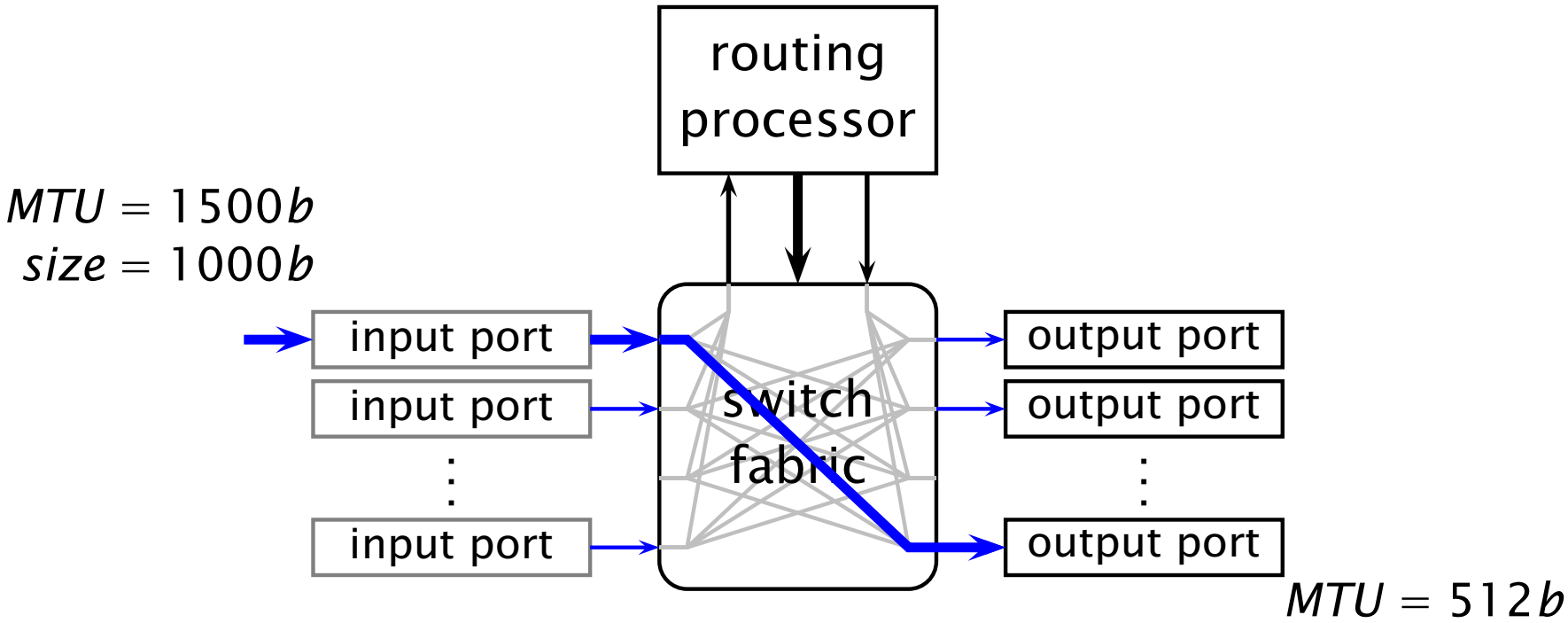
$MTU = 1500b$
$size = 1000b$



Fragmentation

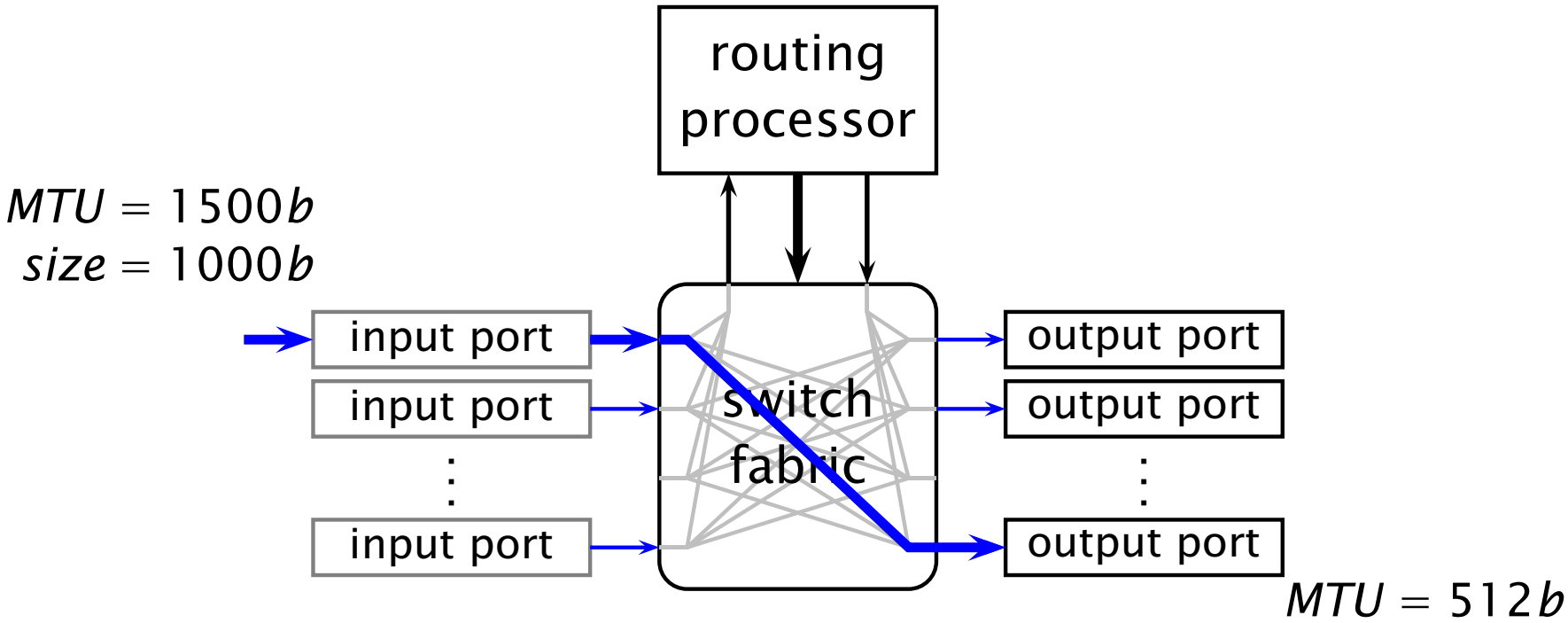


Fragmentation



- How does the router handle cases where the size of an input datagram exceeds the maximum transmission unit (MTU) of the output link?

Fragmentation

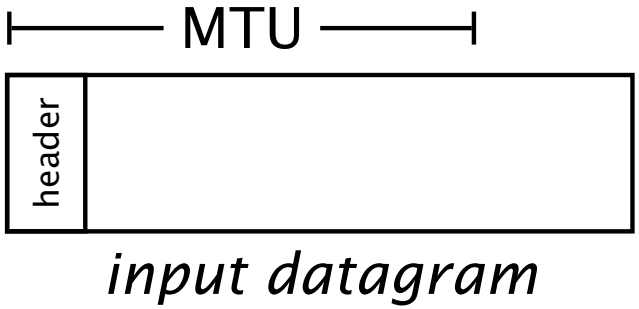


- How does the router handle cases where the size of an input datagram exceeds the maximum transmission unit (MTU) of the output link?
- The datagram is *fragmented*

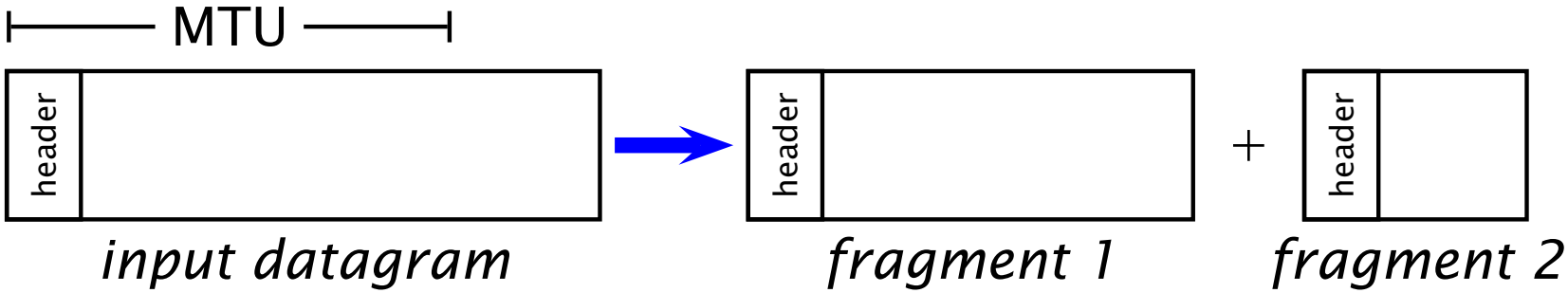
Fragmentation

input datagram

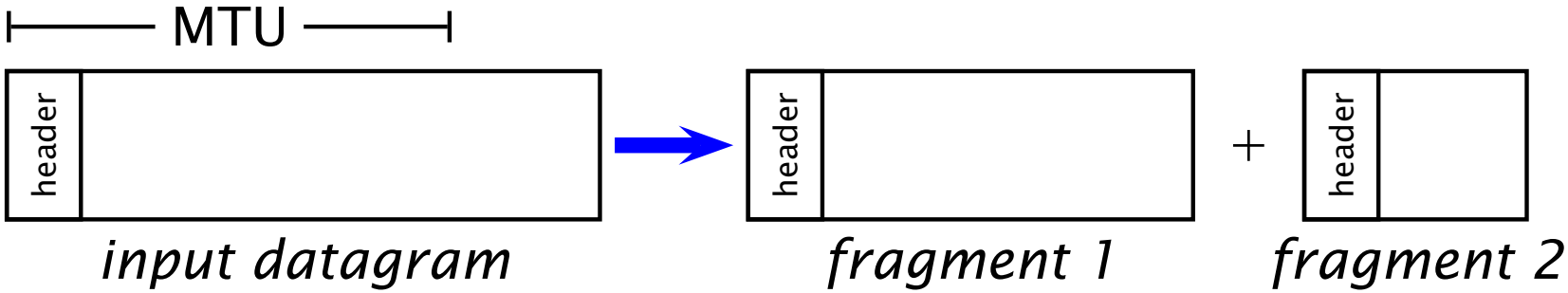
Fragmentation



Fragmentation

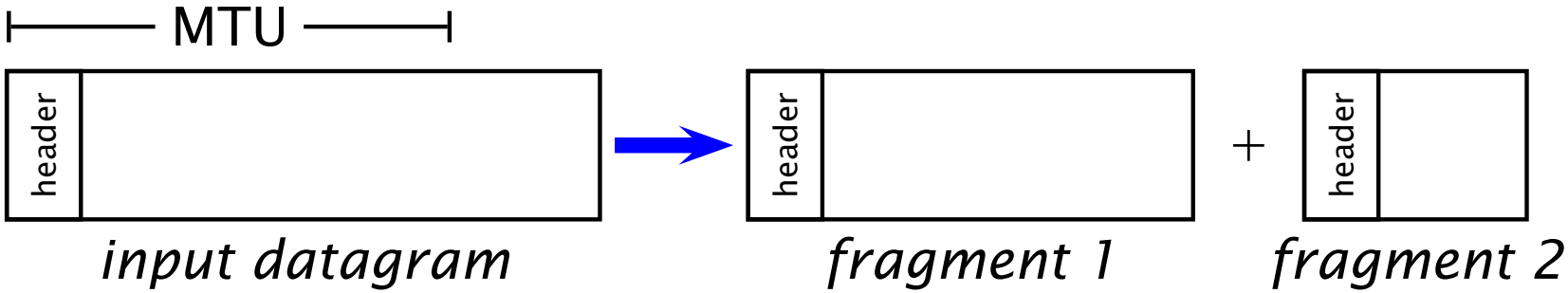


Fragmentation



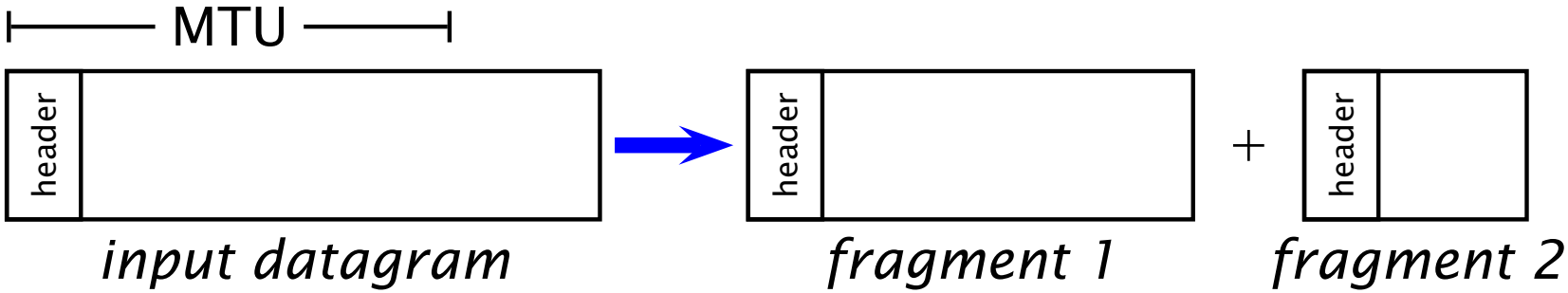
- The *destination* reassembles fragmented datagrams

Fragmentation



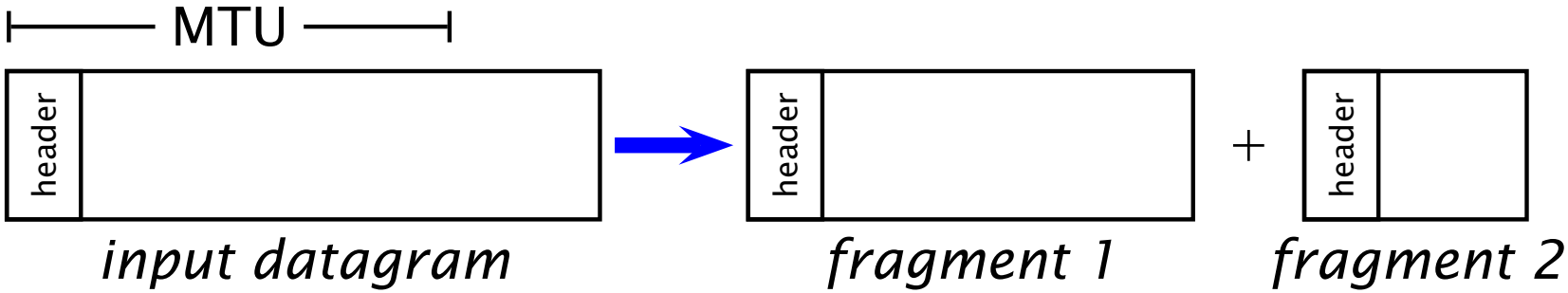
- The *destination* reassembles fragmented datagrams
  - not intermediate routers
  - this is mainly to push complexity out of the network

Fragmentation



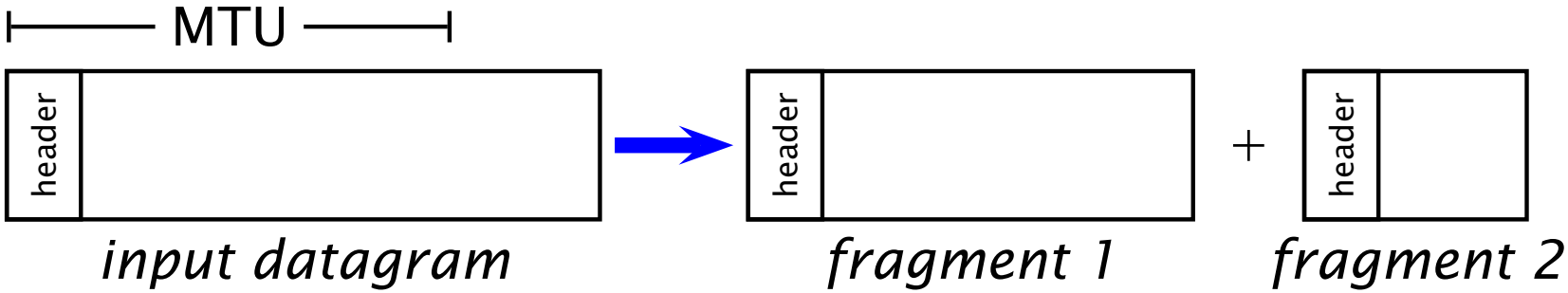
- The *destination* reassembles fragmented datagrams
  - not intermediate routers
  - this is mainly to push complexity out of the network
  - a datagram may have to be fragmented further along the path

Fragmentation



- The *destination* reassembles fragmented datagrams
  - not intermediate routers
  - this is mainly to push complexity out of the network
  - a datagram may have to be fragmented further along the path
- The fragmentation scheme must ensure that the destination host
  - can recognize two fragments of the same original datagram
  - can figure out if and when all the fragments have been received

Fragmentation



- The *destination* reassembles fragmented datagrams
  - not intermediate routers
  - this is mainly to push complexity out of the network
  - a datagram may have to be fragmented further along the path
- The fragmentation scheme must ensure that the destination host
  - can recognize two fragments of the same original datagram
  - can figure out if and when all the fragments have been received
- The fragmentation scheme must ensure that the intermediate routers can fragment a datagram to whatever level necessary

Fragmentation

Fragmentation

- Initial (non-fragmented) datagram format (*datasize* = 1000)

Fragmentation

- Initial (non-fragmented) datagram format (*datasize* = 1000)
  - sender host assigns a 16-bit *identifier* to the datagram (e.g., 789)

Fragmentation

- Initial (non-fragmented) datagram format (*datasize* = 1000)
  - sender host assigns a 16-bit *identifier* to the datagram (e.g., 789)
  - the *fragment offset* is set to 0, indicating that this packet contains data starting at position 0 of the original datagram
    - *fragment offset* is actually the offset in units of 8 bytes (remember it's only 13 bits...)

Fragmentation

- Initial (non-fragmented) datagram format (*datasize* = 1000)
  - sender host assigns a 16-bit *identifier* to the datagram (e.g., 789)
  - the *fragment offset* is set to 0, indicating that this packet contains data starting at position 0 of the original datagram
    - *fragment offset* is actually the offset in units of 8 bytes (remember it's only 13 bits...)
  - the *"more fragments"* flag is set to 0, indicating that no (more) fragments have been sent

Fragmentation

- Initial (non-fragmented) datagram format (*datasize* = 1000)
  - sender host assigns a 16-bit *identifier* to the datagram (e.g., 789)
  - the *fragment offset* is set to 0, indicating that this packet contains data starting at position 0 of the original datagram
    - *fragment offset* is actually the offset in units of 8 bytes (remember it's only 13 bits...)
  - the *"more fragments"* flag is set to 0, indicating that no (more) fragments have been sent

identifier	*fragment offset*	*more fragments*	*header length*	*total length*
789	0	0	20	1020

Fragmentation

Fragmentation

- Fragmentation to an MTU of 512

Fragmentation

- Fragmentation to an MTU of 512
  - sender must split the datagram into 3 fragments:

Fragmentation

- Fragmentation to an MTU of 512
  - sender must split the datagram into 3 fragments:

identifier	*fragment offset*	*more fragments*	*header length*	*total length*
789	0	1	20	508

Fragmentation

- Fragmentation to an MTU of 512
  - sender must split the datagram into 3 fragments:

identifier	*fragment offset*	*more fragments*	*header length*	*total length*
789	0	1	20	508

identifier	*fragment offset*	*more fragments*	*header length*	*total length*
789	61	1	20	508

Fragmentation

- Fragmentation to an MTU of 512
  - sender must split the datagram into 3 fragments:

identifier	*fragment offset*	*more fragments*	*header length*	*total length*
789	0	1	20	508

identifier	*fragment offset*	*more fragments*	*header length*	*total length*
789	61	1	20	508

identifier	*fragment offset*	*more fragments*	*header length*	*total length*
789	122	0	20	44