
DFS 2, Communication Protocols

2 May 2011
Lecture 8

Topics for Today

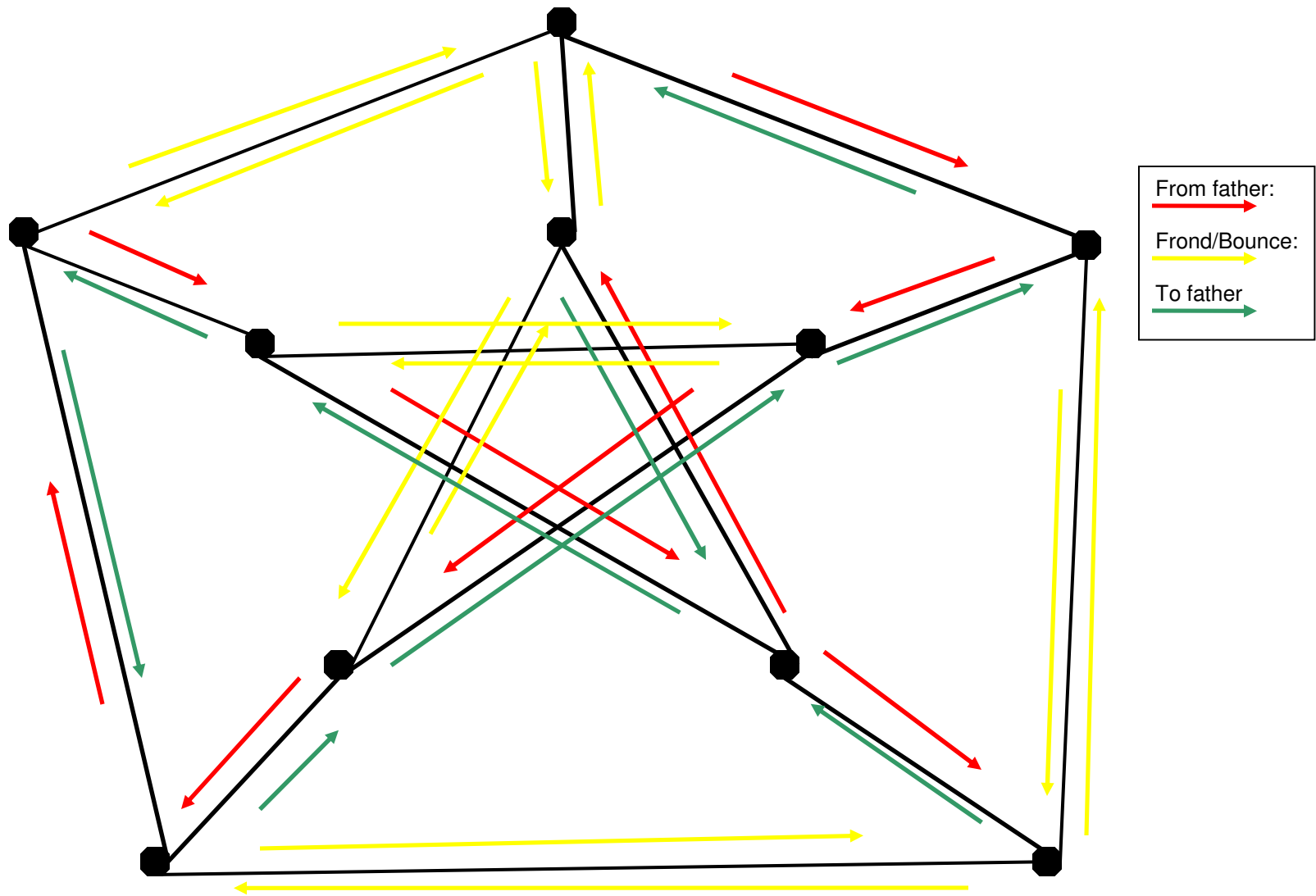
- Depth-first Search
 - Distributed depth first search tree construction
 - Classical
 - Awerbuch
 - Neighbor Knowledge
- Wave and Traversal Algorithms Summary
- Layered network architecture
- Reliable Data Transmission
 - Error models
 - Connection Management
- Balanced Sliding Window Protocol

Source: 6.3, 6.4, 3.1

DFS-algorithms overview

<u>Algorithm</u>	<u>Topology</u>	<u>C/D</u>	<u>Messages</u>	<u>Time</u>
Classic DFS	Any	C	$2 E $	$2 E $
Awerbuch	Any	C	$4 E $	$4 V -2$
Cidon <i>(Skipped)</i>	Any	C	$4 E $	$2 V -2$
Classic DFS + Neighbor Knowledge	Any	C	$2 V -2$	$2 V -2$

Classical DFS on Peterson's graph



Classical DFS-algorithm

```
var     $used_p[q]$  : boolean    init false for each  $q$  in  $Neigh_p$   
       $father_p$  : node          init undef
```

For the initiator execute once:

```
begin   $father_p := p$ ; choose  $q$  in  $Neigh_p$ :  
       $used_p[q] := true$ ; send <tok> to  $q$   
end
```

For each process, upon receipt of <tok> from q_0 :

```
begin  if  $father_p = undef$  then  $father_p := q_0$ ;  
      if for all  $q$  in  $Neigh_p$  :  $used_p[q] = true$   
        then decide  
      else ForwardToken()  
end
```

So far identical to Tarry's traversal algorithm

Classical DFS-algorithm (ctnd)

Forwarding rules:

1. A process never forwards the token twice through the same channel.
2. A non-initiator forwards the token to its father only if there is no other channel left according to rule 1.
3. **When a process receives the token it sends it back through the same channel if this is allowed by rules 1 and 2.**

Classical DFS-algorithm (forwarding)

Procedure *ForwardToken* (p, q_0 : node)

begin

if $\exists q \in Neigh_p : (q \neq father_p \wedge \neg used_p[q])$

then begin

if $father_p \neq q_0 \wedge \neg used_p[q_0]$

then $q := q_0$;

else $q \in \{r \in Neigh_p \mid r \neq father_p \wedge \neg used_p[r]\}$

$used_p[q] := true$; send **<tok>** to q

end

else begin

$used_p[father_p] := true$;

send **<tok>** to $father_p$

end

end

Awerbuch DFS

The **problem** with Classic DFS is that the token runs on each edge **serially**

- That makes the time complexity also relative to the number of edges
- The token passes along each tree edge and each **frond edge**

We can reduce the time complexity by not passing the token over frond edges

- How?

Solution:

- Introduce **receipt acknowledgements**
 - Each node informs all of its neighbors (except its father) when it has received the token for the first time (<vis>/<ack>)
 - Nodes forward when they have received all acks
 - Possible improvement – don't <vis> to the one you will send to

Awerbuch's DFS-algorithm

```
var     $used_p$     : boolean      init false
        $father_p$   : node         init undef;
```

For the initiator execute once:

```
begin  $father_p := p; q \in Neigh_p;$ 
```

```
  forall  $r \in Neigh_p \setminus \{father_p\}$  do send  $\langle vis \rangle$  to  $r$ ;
```

```
  forall  $r \in Neigh_p \setminus \{father_p\}$  do receive  $\langle ack \rangle$  from  $r$ ;
```

```
end     $used_p := true; send \langle tok \rangle$  to  $q$ ;
```

For each process, upon receipt of $\langle tok \rangle$ from q_0 :

```
begin if  $father_p = undef$  then
```

```
  begin  $father_p := q_0$ ;
```

```
    forall  $r \in Neigh_p \setminus \{father_p\}$  do send  $\langle vis \rangle$  to  $r$ ;
```

```
    forall  $r \in Neigh_p \setminus \{father_p\}$  do receive  $\langle ack \rangle$  from  $r$ ;
```

```
  end
```

```
  if  $p$  is initiator  $\wedge \forall q \in Neigh_p : used_p[q]$ 
```

```
  then decide else ForwardToken( $p, q_0$ );
```

```
end
```

Awerbuch's DFS-algorithm (ctnd)

For each process, upon receipt of **<vis>** from q_0
begin $used_p[q_0] = \text{true}$; send **<ack>** to q_0 **end**

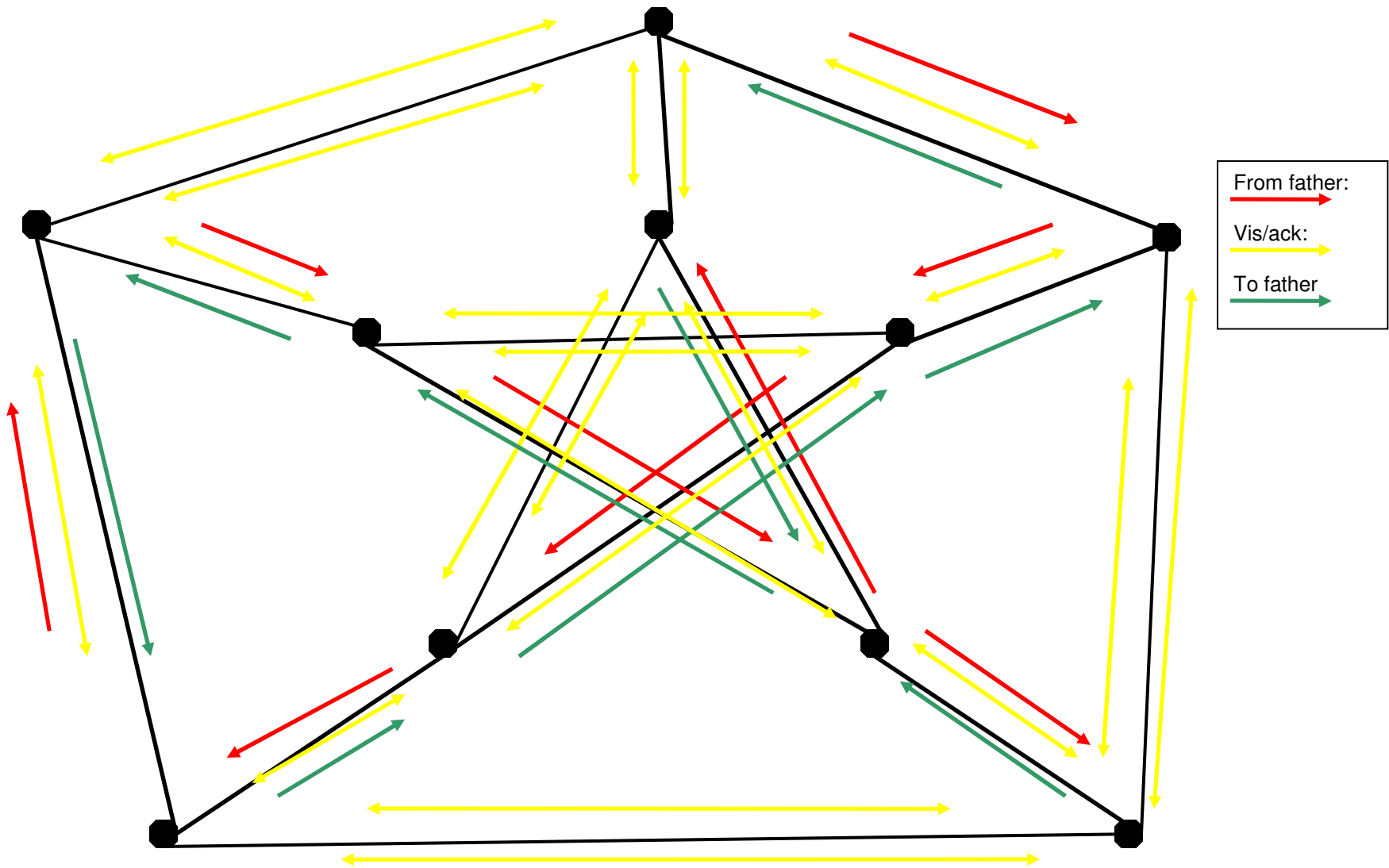
Over each **frond** edge the following messages are sent:

1. One (1) **<vis>** message in each direction.
2. One (1) **<ack>** message in each direction

Over each **tree** edge the following messages are sent:

1. One (1) **<tok>** message in each direction
2. One (1) **<vis>** message from **father to son**
3. One (1) **<ack>** message from **son to father**

Awerbuch DFS on Peterson's graph



DFS with neighbor knowledge

var $father_p$: node *init undef;*

For the initiator only, execute once:

begin $father_p := p$; **choose** q in $Neigh_p$;
 send **<tlist, {p}>** to q
end

For each process, upon receipt of **<tlist, L>** from q_0 :

begin **if** $father_p = undef$ **then** $father_p := q_0$;
 if exists q in $Neigh_p \setminus L$
 then **begin** **choose** q in $Neigh_p \setminus L$;
 send **<tlist, L U {p}>** to q
 end
 else **if** p is initiator
 then *decide*
 else send **<tlist, L U {p}>** to $father_p$
end

Wave and Traversal Summary

- Wave and traversal algorithms deal with networks as abstract *graphs*
 - Hence we began the course with some graph theory
- The algorithms we saw:
 - Wave: Enable computation of common tasks (synchronization, mathematical operations, global acknowledgement)
 - Traversal: efficiently build subgraphs with known properties
- Foundations for many common problems you may face in designing algorithms in computer networks

So Far

- Depth-first Search
 - Distributed depth first search tree construction
 - Classical
 - Awerbuch
 - Neighbor Knowledge
- Wave and Traversal Algorithms Summary
- Layered network architecture
- Reliable Data Transmission
 - Error models
 - Connection Management
- Balanced Sliding Window Protocol

OSI Model, TCP/IP Model

OSI	TCP/IP
Application	Application
Presentation	
Session	Transport
Transport	
Network	Network
Data Link	Physical
Physical	

Usually the data link OSI-layer is subdivided into two layers:

1. MAC layer
2. Logical link layer

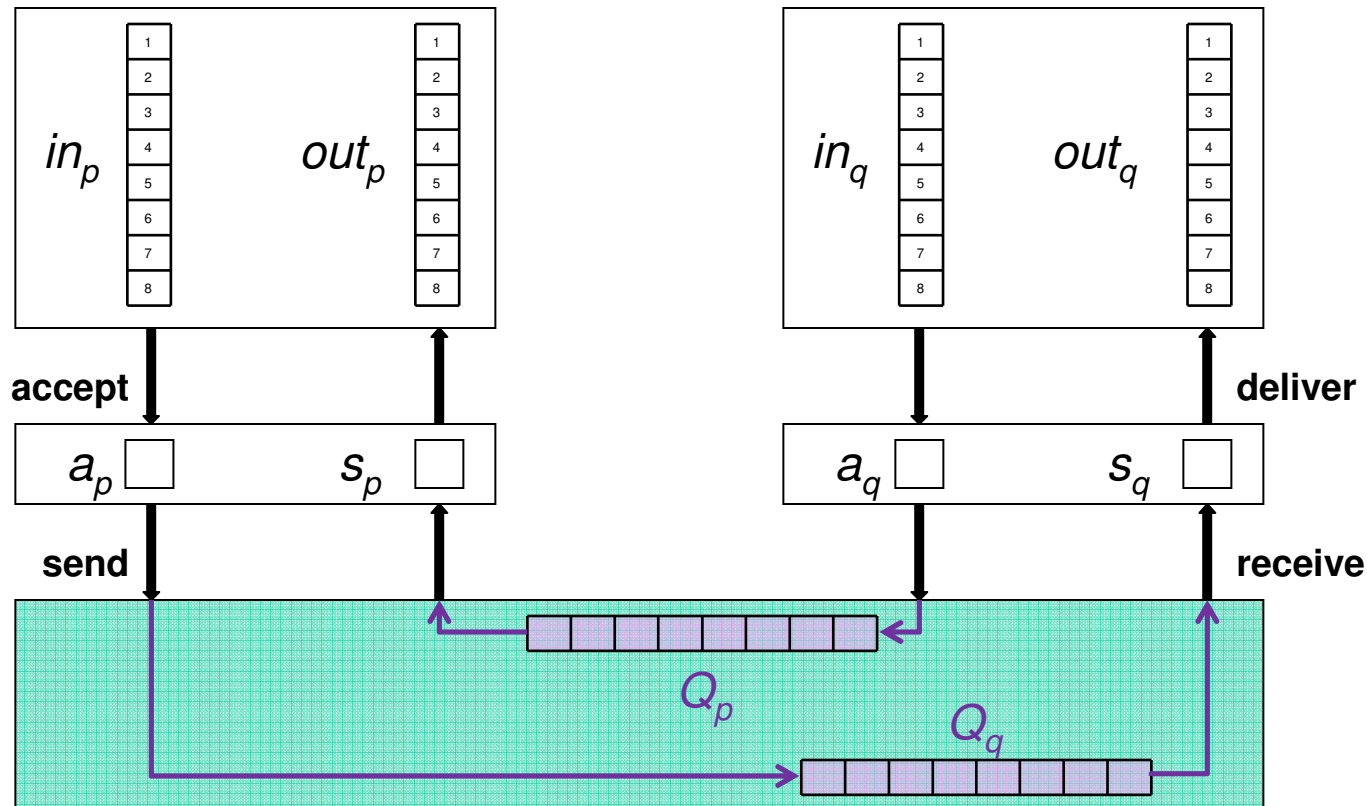
Aspects of Communication Protocols

- Purpose
 - reliable data transmission
 - may be implemented by reporting transmission errors
- OSI layer
 - Single-hop connection (link layer)
 - deals with transmission errors due to **physical circumstances**
 - Multi-hop connection (transport layer)
 - deals with errors caused by **limited resources and protocol actions on intermediate machines**
- Characteristics
 - Error model
 - loss, duplication, garbling, creation
 - State information
 - Timing assumptions

Balanced Sliding-Window Protocol

- Assumptions
 - Processes are directly connected (data-link layer protocol)
 - or virtual channel on transport layer (TCP)
 - **bidirectional** connections
 - error model: **loss** (garbling = loss)
 - FIFO property:
 - channel pq is modeled by a pair of queues Q_p and Q_q
- Requirements
 - Safety
 - all delivered data is correct
 - Liveness
 - all data is eventually delivered

Protocol Model



1. in_p : infinite array of words to be delivered by p
2. out_p : infinite array of words to be accepted by p
3. a_p : the lowest numbered word not yet acknowledged by q
4. s_p : the lowest numbered word not yet received from q

Conclusion

- Depth-first Search
 - Distributed depth first search tree construction
 - Classical
 - Awerbuch
 - Neighbor Knowledge
- Wave and Traversal Algorithms Summary
- Layered network architecture
- Reliable Data Transmission
 - Error models
 - Connection Management
- Balanced Sliding Window Protocol