
Timer Based Sliding Window, Routing Algorithms

26 May 2010
Lecture 10

Slide credits: R. H. Mak

Topics for Today

- Timer-based Sliding Window Protocol
- Routing
 - Routing on the Internet
 - Routing Criteria and Cost Metrics
 - Destination-based Routing
 - All-pairs Shortest Path
 - Floyd-Warshall
 - Routing Algorithms
 - Toueg's Algorithm
 - Chandy-Misra Algorithm
 - Netchange Algorithm
 - Table Compaction

Source: Tel 3.3, 4.1, 4.2

Timer-based protocol

- End-to-end protocol (transport protocol)
- Connection management
- Resilient against
 - Loss, duplication, reordering errors
- Ordered delivery within bounded time or reported loss
- Simplified Fletcher Watson Δt -protocol
 - One direction (data flow)
 - Single-word packets
 - Single-word receiving window
 - Simplified timing assumptions

Time and timer assumptions

- Global time
 - Each event takes place at a certain moment in global time, not observable by the processes
 - Each event is instantaneous (has duration zero)
 - The time event included!?
- Bounded packet life time
 - A packet sent at $\tau = \sigma$ and any of its duplicates, must be received before $\tau = \sigma + \mu$ or becomes lost
- Timers
 - Processes cannot observe absolute time, but have access to timers $Xt^{(t_1)} - Xt^{(t_2)} = t_2 - t_1$

Protocol Description

1. A connection is opened
 - by the sender, when a word is accepted and no connection exists
 - by the receiver upon receipt of a word, when no connection exists
2. A connection is closed
 - by the sender, upon time-out after the last send action, provided all packages sent have been acknowledged (or reported lost, see point 7)
 - by the receiver, upon time-out after the last delivery of a word
3. A packet is lost in transmission
 - when the packet life time is exceeded
4. A packet is duplicated
 - At a random moment during transmission
 - A duplicate has the same remaining life time as the original!

Protocol Description

5. Packet (re)transmission

1. The sender keeps track of the number of words accepted and delivered during the connection.
2. The sender (re-)transmits an accepted word that is not yet delivered only during a finite time interval after its acceptance.
3. The receiver keeps track of the lowest numbered that has not been delivered in the current connection.
4. The receiver discards (does not deliver) words that arrive out of order.
5. The receiver repeatedly notifies the sender of its last delivery (by sending acknowledgements) as long as the connection remains open.

6. Timer values are chosen such that an acknowledgement cannot be received when a connection is closed

Protocol Description

7. Before a connection is closed all accepted words that have not been delivered are reported lost to the producer.
 - The producer must decide whether it wants to open a new connection to retransmit the words that have not been delivered in previous connection.

Constants and Network State

Network Constant:

μ : real; (* Maximum packet lifetime *)

Protocol Constants:

U : real; (* Length of send interval *)

R : real; (* Receiver time-out value: $R \geq U + \mu$
no duplicates can arrive *)

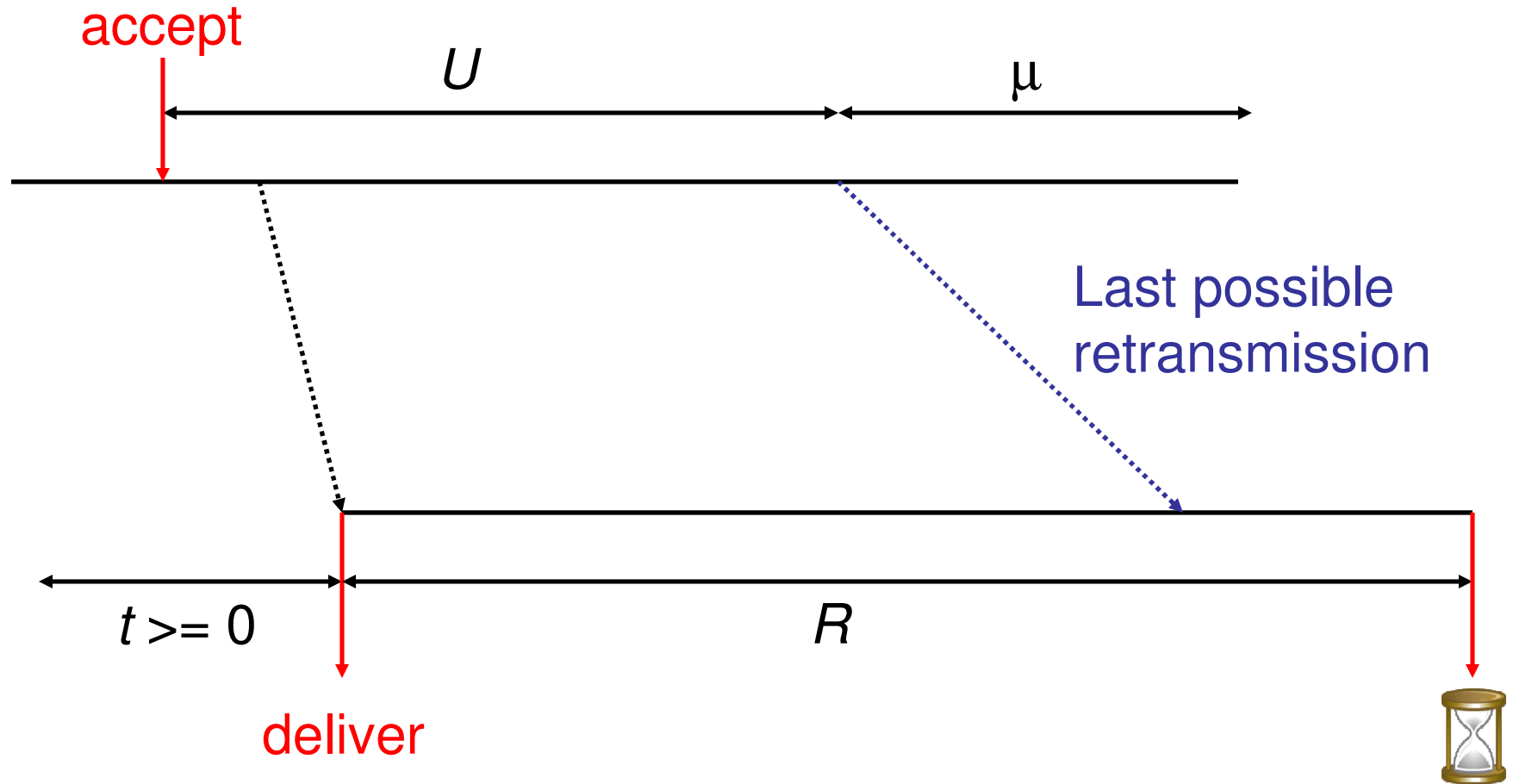
S : real; (* Sender time-out value: $S \geq R + 2\mu$
no acknowledgement can arrive *)

Communication Subsystem;

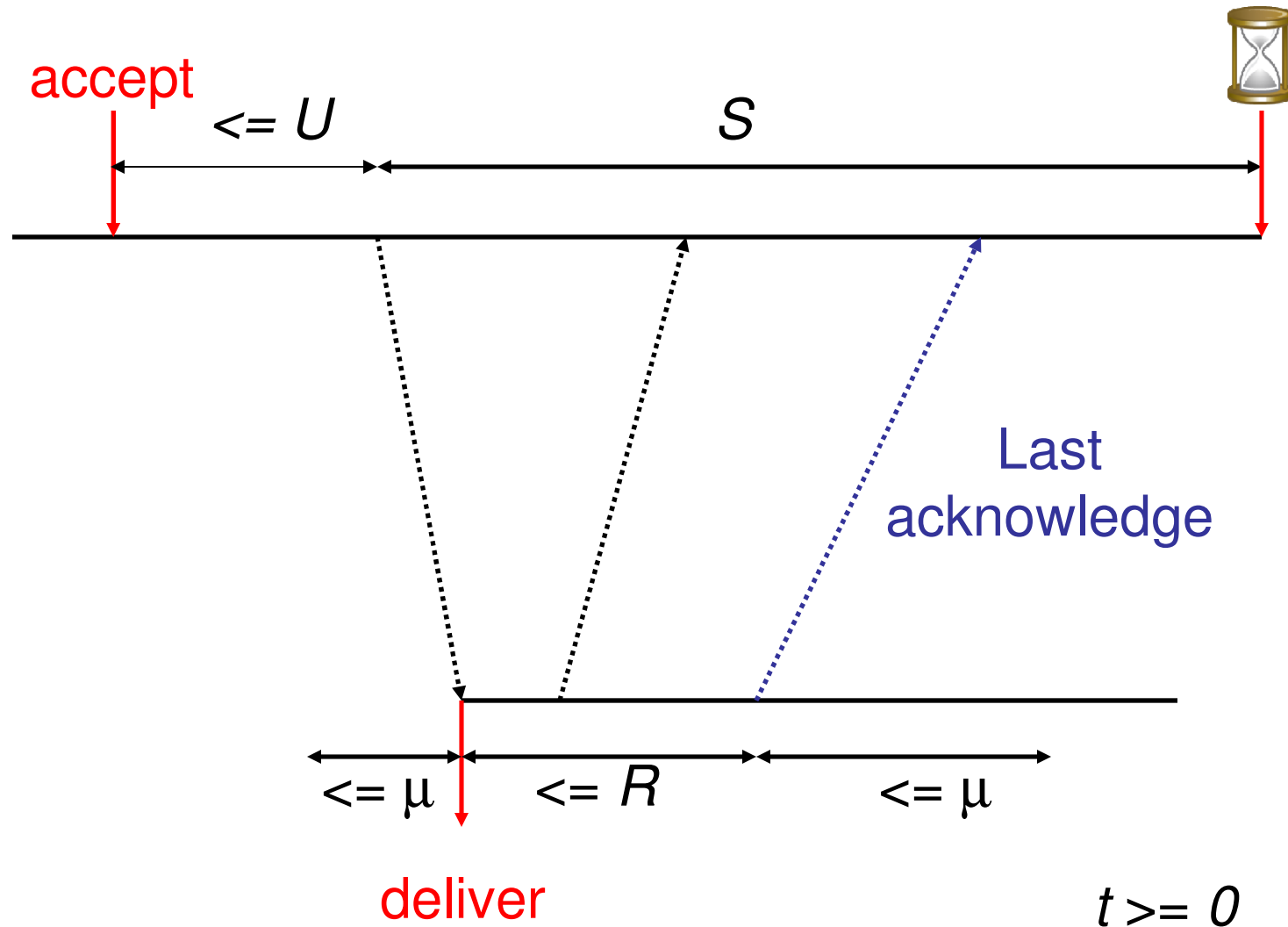
M_q : channel; (* Data packets for q *)

M_p : channel; (* Acknowledge packet for p *)

Reason for $R \geq U + \mu$



Reason for $S \geq R + 2\mu$



Process State

Connection record of sender:

Low : int; (* Acknowledged words current connection *)

High : int; (* Accepted words current connection *)

St : timer (* Connection timer *)

Connection record of receiver:

Exp : int; (* Next expected sequence number *)

Rt : timer; (* Connection timer *)

Auxiliary variables:

B : int **init** 0; (* Words in previous connections *)

cr : boolean **init** *false* (* Connection exists at receiver *)

cs : boolean **init** *false* (* Connection exists at sender *)

Ut [*i*] : timer; (* For all indices $Low \leq i < High$ *)

Sender Protocol (i)

A_p : (* Accept next word *)

begin if !cs then

begin (* Connection is opened first *)

create (St , $High$, Low); (* $cs := true$ *)

$Low := High := 0$; $St := S$;

end;

$Ut [B + High] := U$; $High := High + 1$;

end;

S_p : (* Send i^{th} word from current connection *)

{ $cs \wedge Low \leq i < High \wedge Ut [B + i] > 0$ }

begin send <data, ($i = Low$), i , $in_p [B + i]$, μ >;

$St := S$;

end;

Sender Protocol (ii)

R_p : (* Receive and acknowledgement *)

$\{cs \wedge \langle \mathbf{ack}, i, \rho \rangle \in M_p\}$

begin receive $\langle \mathbf{ack}, i, \rho \rangle$; $Low := \max(Low, i)$ **end**;

E_p : (* Generate error report for possibly lost word*)

$\{cs \wedge Ut [B + Low] < -2\mu - R\}$

begin error $[B + Low] := true$; $Low := Low + 1$ **end**;

C_p : (* Close the connection *)

$\{cs \wedge St < 0 \wedge Low = High\}$

begin $B := B + High$; delete $(St, High, Low)$ **end**;

(* $cs := false$ *)

Receiver Protocol

R_q : (* Receive a data packet *)
 {<data, s , i , w , ρ > $\in M_q$ }
 begin receive <data, s , i , w , ρ >;
 if cr **then**
 if $i = Exp$ **then**
 begin $Rt := R$; $Exp := i+1$; deliver w **end**;
 else if $s = true$ **then**
 begin create < Rt , Exp >; (* $cr := true$ *)
 $Rt := R$; $Exp := i + 1$; deliver w
 end
 end

S_q : (* Send an acknowledgement *)
 { cr }
 begin send <ack, Exp , μ > **end**;

(* Close connection if Rt times out; see action **Time** *)

Channel and Time Actions

Loss : $\{m \in M\}$ (* M is either M_p or M_q *)
 begin remove m from M ; **end**;

Dupl: $\{m \in M\}$ (* M is either M_p or M_q *)
 begin insert m in M ; **end**;

Time : (* $\delta > 0$ *)
 begin forall i **do** $Ut[i] := Ut[i] - \delta$;
 $St := St - \delta$;
 $Rt := Rt - \delta$;
 if $Rt \leq 0$ **then** delete (Rt, Exp) ; (* $cr := false$ *)
 forall $\langle \dots, \rho \rangle \in M_p, M_q$ **do**
 begin $\rho := \rho - \delta$;
 if $\rho \leq 0$ **then** remove packet
 end
 end

So Far

- Timer-based Sliding Window Protocol
- Routing
 - Routing on the Internet
 - Routing Criteria and Cost Metrics
 - Destination-based Routing
 - All-pairs Shortest Path
 - Floyd-Warshall
 - Routing Algorithms
 - Toueg's Algorithm
 - Chandy-Misra Algorithm
 - Netchange Algorithm
 - Table Compaction

Routing protocols on the internet

- Treated extensively in ISE 327, EEE 051 Introduction to Computer Networks
- Routing information protocol (RIP)
 - Distance vector routing
 - Uses Bellman-Ford algorithm
 - Outdated, suffers from count-to-infinity problem
- Open shortest path first (OSPF)
 - Link state routing
 - Uses Dijkstra algorithm to determine shortest paths
- Intermediate system to intermediate system IS-IS
 - Similar to OSPF
- Border gateway protocol (BGP)
 - Between networks (administrative domains, autonomous systems)
 - Path-vector routing
 - (We didn't get to it in ISE 327, but read about it on [Wikipedia](#) if you want)

Routing criteria

- Correctness
 - Every packet is delivered at its destination
- Efficiency
 - Paths with small delay and high throughput (network wide)
- Complexity
 - Setting up routing tables, making routing decision
- Robustness
 - Cope with topology changes, no network reboot
- Adaptiveness
 - Load balancing and traffic control
- Fairness
 - all users get the same degree of service

Path costs, routing metrics

- Minimum hop
 - Number of channels traversed
- Shortest path
 - Channels have statically assigned weights and the cost of a path is the sum of costs of the edges
 - presumes no negative-cost cycles
- Minimum delay
 - Channels have dynamically assigned weights based on the traffic on the channel
 - Routing tables are repeatedly revised such that paths with close to minimal delays are chosen

Destination-based routing

Decision where to forward a packet depends only on the destination of a packet (not on its source)

Optimal paths can be determined, when

1. the cost of sending a packet via a path is independent of the amount of traffic on the path.
2. the cost of the concatenation of two paths equals the sum of the costs of the concatenated paths.
3. The communication network does not contain cycles with negative cost.

Destination-based routing

Lemma. Let $u, v \in V$. If a path from u to v exists in G , then there exists an optimal simple path from u to v .

Theorem. For each $d \in V$ there exists a spanning tree $T_d = (V, E_d)$ such that for each node $v \in V$, the path from v to d in T_d is an optimal path from v to d in G .

Definition. A spanning tree rooted towards d is called a *sink tree* for d .

All-pair shortest path algorithms

1. Floyd-Warshall
 - Sequential algorithm
2. Toueg's algorithm
 - Distributed version of Floyd-Warshall
3. Merlin-Segall
4. Chandy-Misra
5. Netchange (Tajibnapis)

S-paths and S-distance

Let $G=(V,E,\omega)$ be a weighted graph and let $S \subseteq V$. A path $\langle u_0, \dots, u_k \rangle$ is an S -path if all internal nodes belong to S , i.e., $u_i \in S$ for all $0 < i < k$. For $u, v \in V$ the S -distance $d^S(u,v)$ is the lowest weight of an S -path.

If G does not contain cycles with negative weight, then

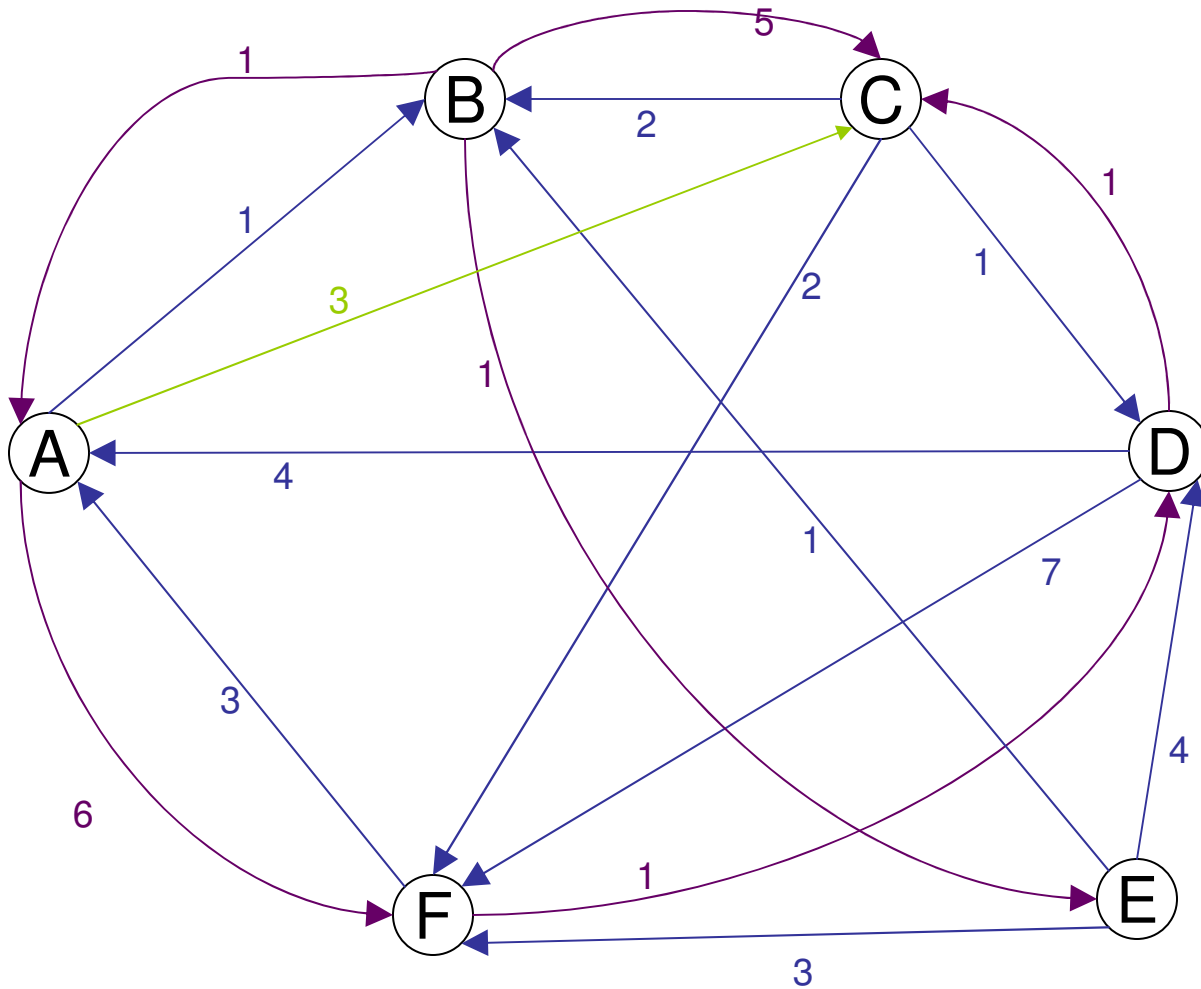
$$d^S(u,u) = 0$$

$$d^0(u,v) = \begin{cases} \omega_{uv}, uv \in E \\ \infty, uv \notin E \end{cases}$$

$$d^{S \cup \{w\}}(u,v) = \min(d^S(u,v), d^S(u,w) + d^S(w,v))$$

$$d(u,v) = d^V(u,v)$$

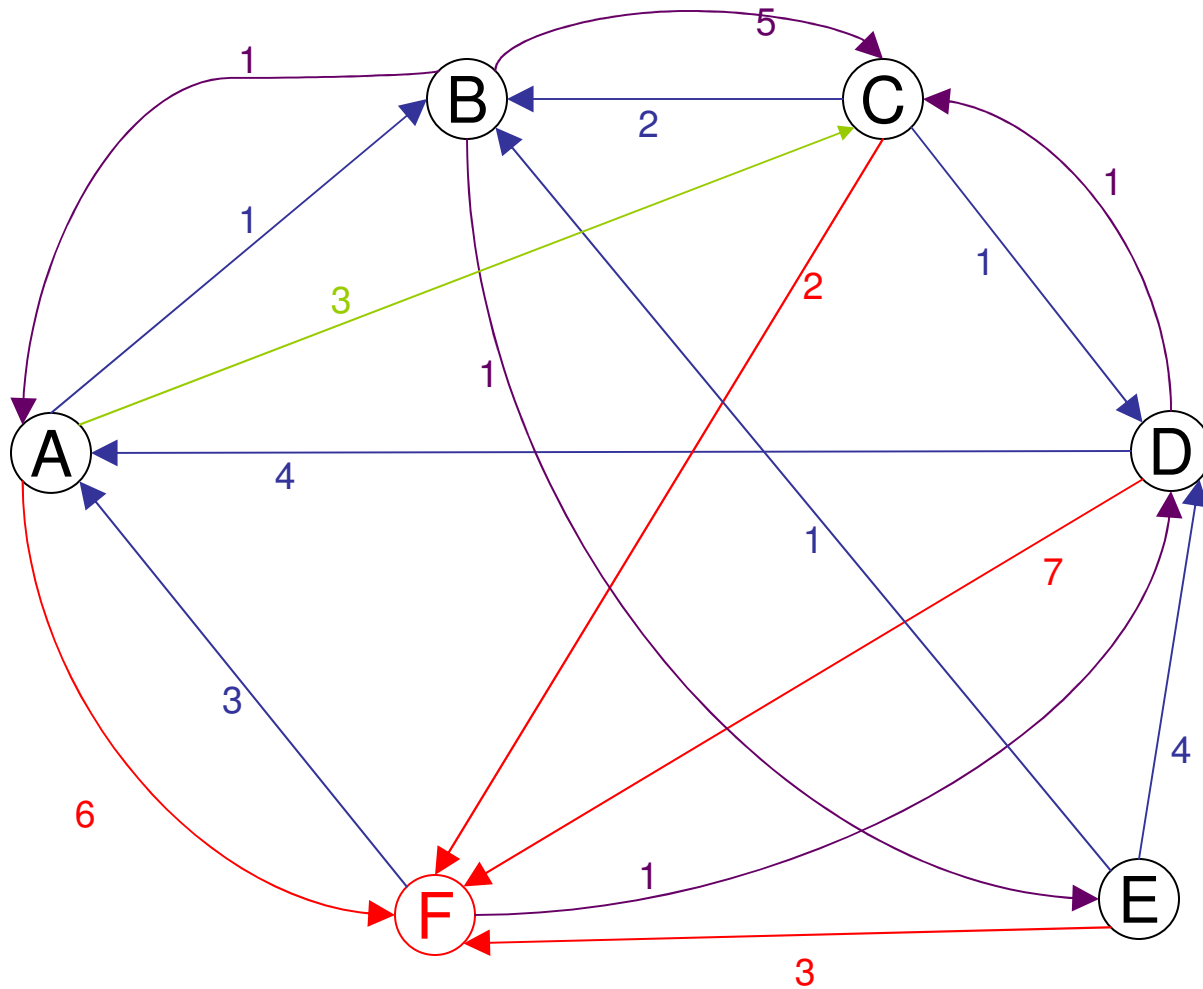
Weighted Graph



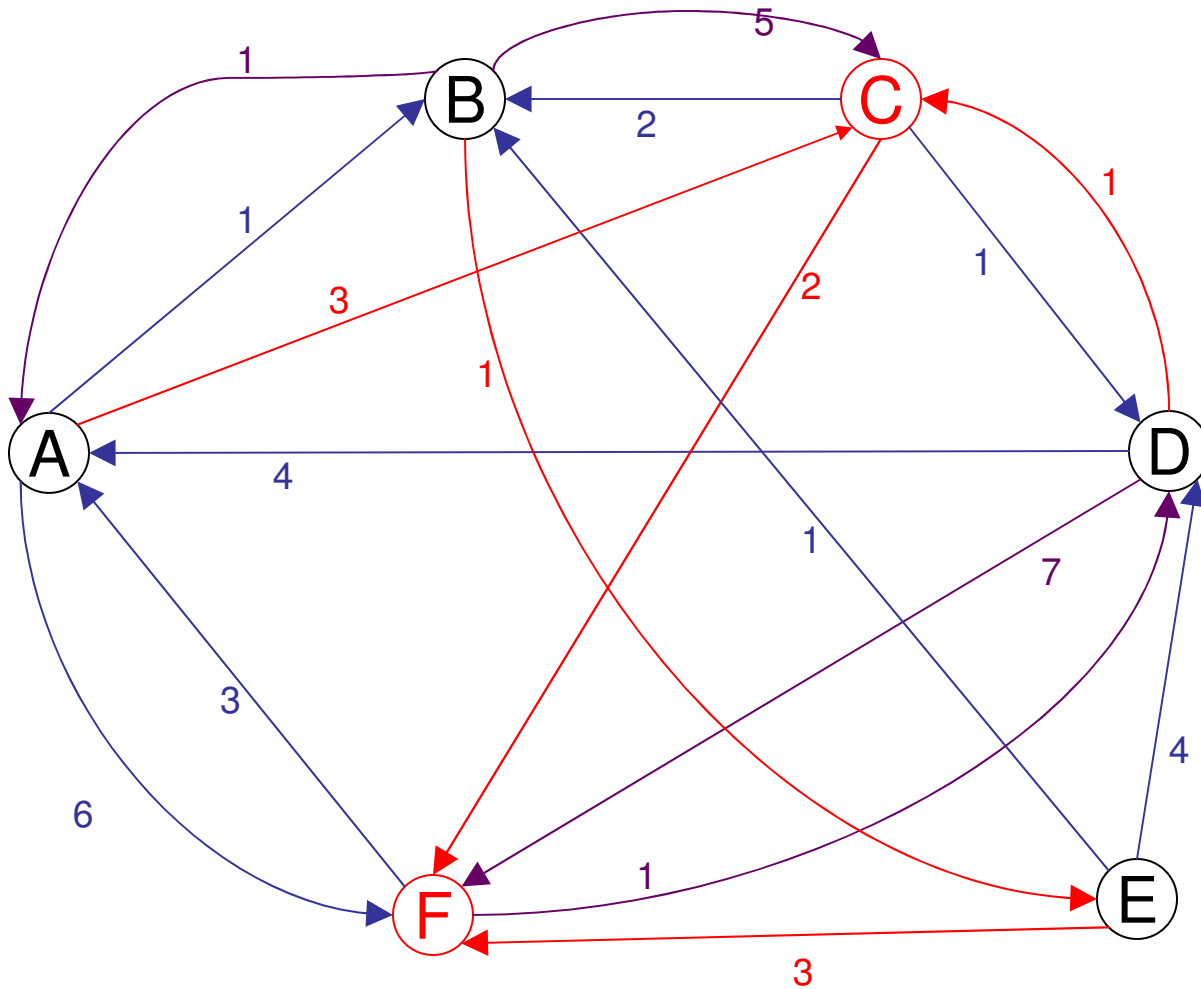
d^0

	A	B	C	D	E	F
A	0	1	3	Inf	Inf	6
B	1	0	5	Inf	1	Inf
C	Inf	2	0	1	Inf	2
D	4	Inf	1	0	Inf	7
E	Inf	1	Inf	4	0	3
F	3	Inf	Inf	1	Inf	0

Optimal Sink Tree Construction



Optimal Sink Tree for F



Adding A, B, C, D, E

	A	B	C	D	E	F
A	0	1	3	Inf	Inf	6
B	1	0	5	Inf	1	Inf
C	Inf	2	0	1	Inf	2
D	4	Inf	1	0	Inf	7
E	Inf	1	Inf	4	0	3
F	3	Inf	Inf	1	Inf	0

	A	B	C	D	E	F
A	0	1	3	Inf	Inf	6
B	1	0	4	Inf	1	7
C	Inf	2	0	1	Inf	2
D	4	5	1	0	Inf	7
E	Inf	1	Inf	4	0	3
F	3	4	6	1	Inf	0

	A	B	C	D	E	F
A	0	1	3	Inf	2	6
B	1	0	4	Inf	1	7
C	3	2	0	1	3	2
D	4	5	1	0	6	7
E	2	1	5	4	0	3
F	3	4	6	1	5	0

Add A

Add B

	A	B	C	D	E	F
A	0	1	3	4	2	5
B	1	0	4	5	1	6
C	3	2	0	1	3	2
D	4	3	1	0	4	3
E	2	1	5	4	0	3
F	3	4	6	1	5	0

	A	B	C	D	E	F
A	0	1	3	4	2	5
B	1	0	4	5	1	6
C	3	2	0	1	3	2
D	4	3	1	0	4	3
E	2	1	5	4	0	3
F	3	4	2	1	5	0

	A	B	C	D	E	F
A	0	1	3	4	2	5
B	1	0	4	5	1	4
C	3	2	0	1	3	2
D	4	3	1	0	4	3
E	2	1	5	4	0	3
F	3	4	2	1	5	0

Add C

Add D

Add E

Adding A, B, C, D, E, F

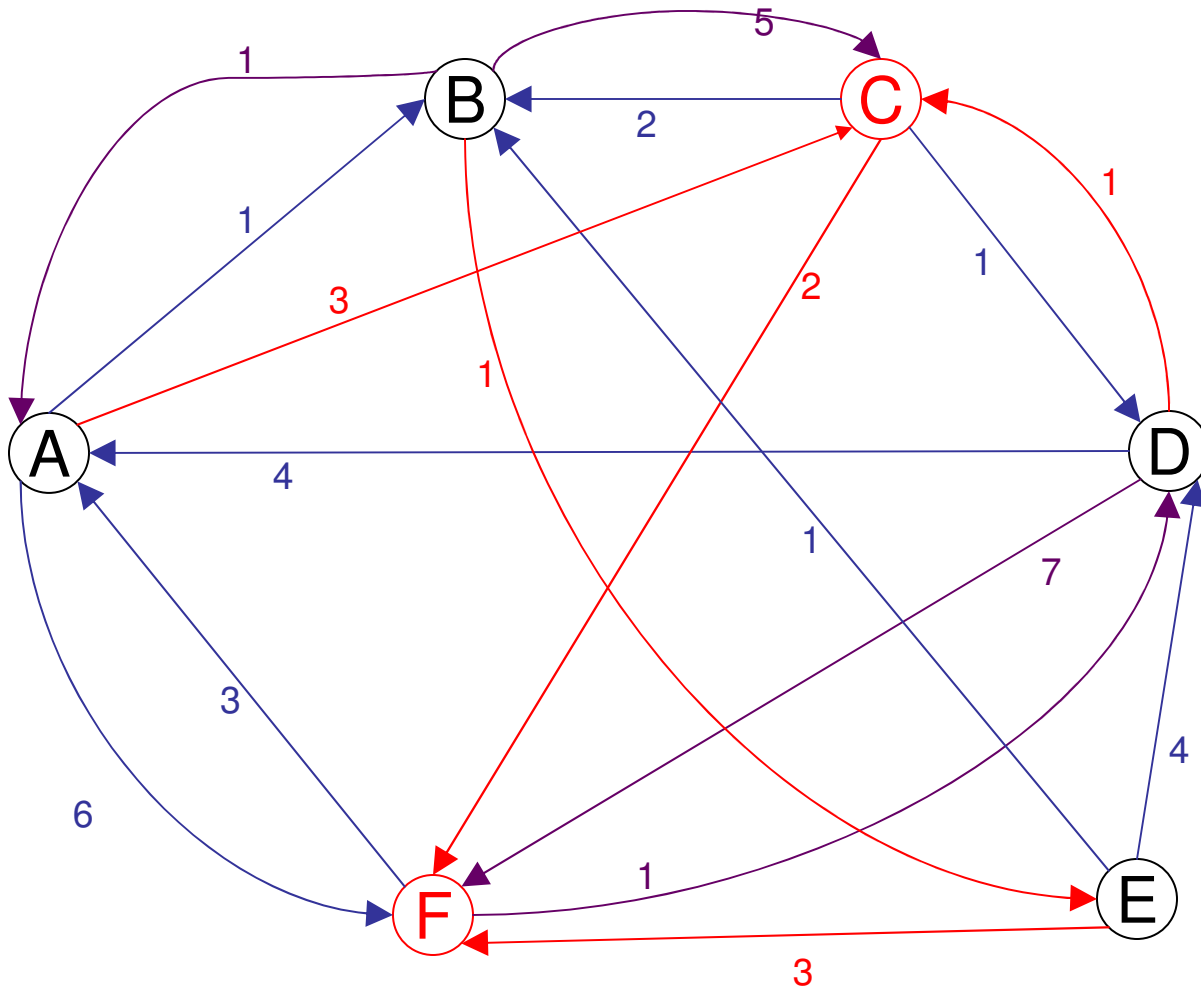
	A	B	C	D	E	F
A	0	1	3	4	2	5
B	1	0	4	5	1	4
C	3	2	0	1	3	2
D	4	3	1	0	4	3
E	2	1	5	4	0	3
F	3	4	2	1	5	0

Add E

	A	B	C	D	E	F
A	0	1	3	4	2	5
B	1	0	4	5	1	4
C	3	2	0	1	3	2
D	4	3	1	0	4	3
E	2	1	5	4	0	3
F	3	4	2	1	5	0

Add F

Optimal Sink Tree



Floyd-Warshall

```
begin S := {};  
  forall u, v do  
    if u = v then D[u,v] := 0;  
    else if uv ∈ E then D[u,v] := ωuv  
         else D[u,v] := Inf;  
  
  while S <> V do  
    (* inv. forall u,v : D[u,v] = dS(u,v)*)  
    begin pick w from V / S;  
      forall u ∈ V do  
        forall v ∈ V do  
          D[u,v] := min(D[u,v], D[u,w] + D[w,v]);  
        S := S U {w};  
      end  
    end  
end
```

Conclusion

- Timer-based Sliding Window Protocol
- Routing
 - Routing on the Internet
 - Routing Criteria and Cost Metrics
 - Destination-based Routing
 - All-pairs Shortest Path
 - Floyd-Warshall
 - Routing Algorithms
 - Toueg's Algorithm
 - Chandy-Misra Algorithm
 - Netchange Algorithm
 - Table Compaction