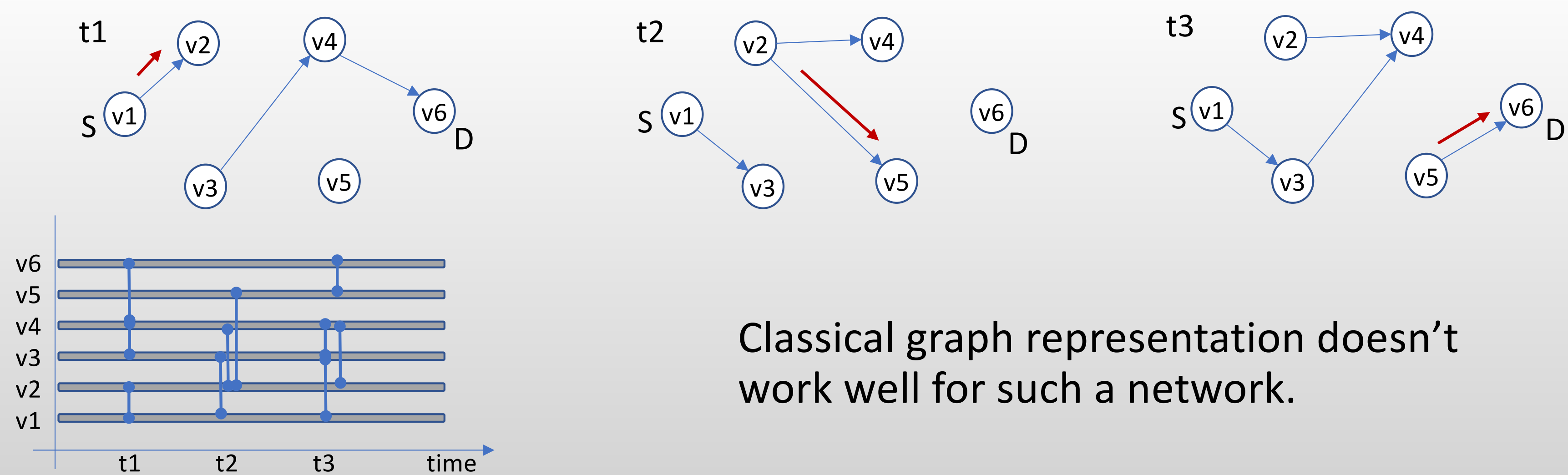
ENPM 809X

E. Gunduzhan

Project 2: Contact Graphs

Need for a Different Representation

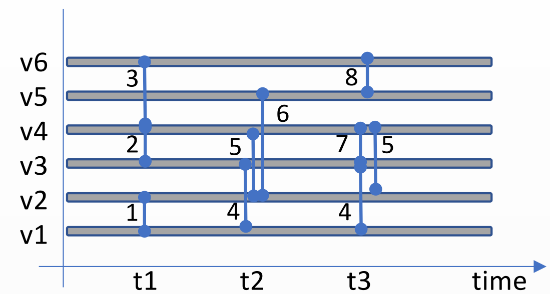
- In a network with mobile nodes, the vertices of a graph keep moving. Links are established or abolished over time. Neighbors change.
- A path from a source S to a destination D may never exist, but it may still be possible to go from S to D in multiple "hops" with possible delay after each hop.

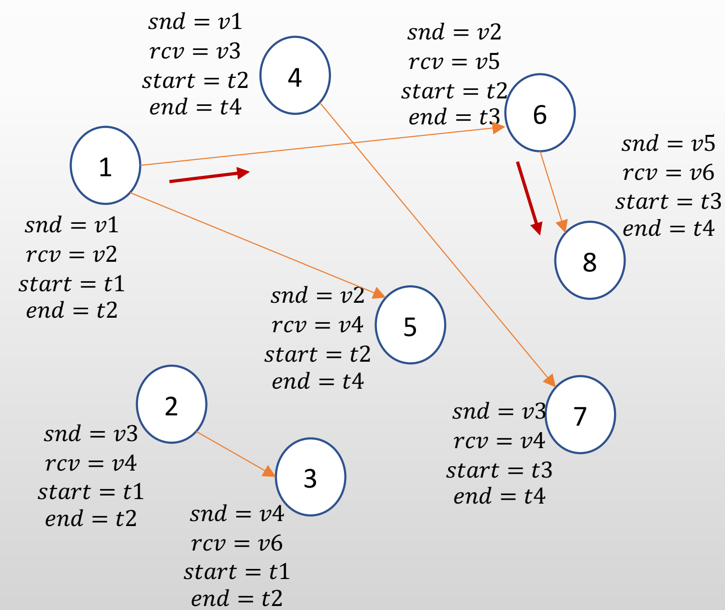


Classical graph representation doesn't work well for such a network.

Contact Graph

- A contact graph $G = (C, E)$ can be defined as follows:
- Each vertex $c_i \in C$ represents a contact between two nodes
  - Sender and receiver nodes of each contact are stored in two attributes of a vertex: $c_i.snd$ and $c_i.rcv$
  - The start and end time of each contact are also stored in two attributes of a vertex: $c_i.start$ and $c_i.end$
  - The transmission delay of a contact is stored in the $c_i.owlt$ attribute
- An edge between two vertices c_i and c_j represents a wait before the next contact is used
  - An edge between c_i and c_j is possible only if $c_i.rcv = c_j.snd$ and $c_i.start > c_j.end$





Earliest Time Problem

- Given a source node S, a destination node D, and a contact graph G, first add a root contact to G such that:

$$C_{root}.snd = C_{root}.rcv = S \qquad C_{root}.start = 0 \qquad C_{root}.end = \infty$$

- Set the arrival time of the root contact $C_{root}.arr\_time$ equal to the current time
- A path in the contact graph G from S to D is a sequence of vertices (contacts) starting with $c_1 = C_{root}$ and ending with c_n, such that:
  - $c_{i+1}.snd = c_i.rcv$
  - $c_n.rcv = D$
  - $c_{i+1}.end \geq c_i.start$

- The arrival time at each vertex along a path is related to the previous arrival time as:

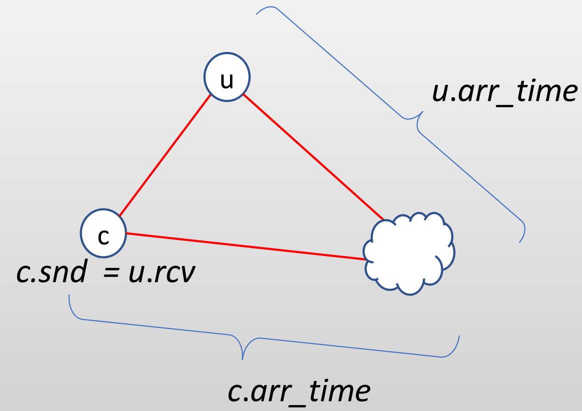
$$\underbrace{c_{i+1}.arr\_time}_{\text{Time message arrives at the receiver of } c_{i+1}} = \underbrace{\max(c_i.arr\_time, c_{i+1}.start)}_{\text{Earliest time the sender of } c_{i+1} \text{ can start sending}} + \underbrace{c_{i+1}.owlt}_{\text{Transmission delay}}$$

- We want to find a path from S to D with the earliest arrival time at D.

Relaxation in CGR

- For each vertex c, maintain an attribute $c.arr\_time$, which is an upper bound on the shortest time from S to c
- Initialize $c.arr\_time = \infty$ and $c.pred$ = NIL
- Relaxation is testing if $c.arr\_time$ would be improved by going through u, and if so, modifying these attributes
- Also maintain an additional attribute, $c.visited\_n$, which is the set of nodes visited in the path from S to c

$arr\_time = \max(u.arr\_time, c.start) + c.owlt$
if $arr\_time < c.arr\_time$
    $c.arr\_time = arr\_time$
    $c.pred = u$



Dijsktra for CGR

CGR(*G, Croot, Cdest*)

BDT = ∞

Cfin = NULL

Ccurr = Croot

```
while 1
    (Cfin, BDT) = CRP(G, Ccurr, Cfin, BDT)

    Ccurr = CSP(G, BDT)

    if Ccurr == NULL
        break
Construct and return the route and time
```

CRP: Visit every valid “neighbor” contact of Ccurr and relax it

CSP: Select the unvisited contact with the least arrival time

Contact attributes to track the shortest paths:

visited : flag to distinguish between visited and unvisited contacts

visited_n : set of visited nodes along the path

pred : predecessor contact

arr_time : upper bound on the shortest time

CRP Function

CRP(*G, Ccurr, Cfin, BDT*)

```
for each  C ∈ G
        if C.src <> Ccurr.dst  or ...
        C.end < Ccurr.arr_time  or ...
        C.visited  or ...
        C.dst ∈ Ccurr.visited_n
                skip C
        if C.start < Ccurr.arr_time
                arr_time = Ccurr.arr_time + C.owlt
        else
                arr_time = C.start + C.owlt
        if arr_time < C.arr_time
                C.arr_time = arr_time
                 C.pred = Ccurr
                 C.visited_n = Ccurr.visited_n ∪ C.dst
                if C.dst == Cdest and C.arr_time < BDT
                                BDT = C.arr_time
                                 Cfin = C
Ccurr.visited = TRUE
return (Cfin, BDT)
```

Only consider the unvisited contacts with matching source, not ending before current's arrival time, and with a destination not visited in current's path

Calculate the arrival time in *C* via *Ccurr*

Relax (also update the visited nodes list)

If *C* reaches the ultimate destination, compare its arrival time against the best arrival time and update if better

CSP Function

CSP(G, BDT)

$Ccurr$ = NULL

$best\_arr = \infty$

for each $C \in G$

    if $C.arr\_time > BDT$ or $C.visited$

        skip C

Only consider the unvisited contacts with arrival time less than *BDT* (no need to consider those with larger arrival times)

    if $C.arr\_time < best\_arr$

        $best\_arr = C.arr\_time$

        $Ccurr = C$

Find and return such a contact with the smallest arrival time

return ($Ccurr$)

- Uses linear search for the minimum arrival time
- Instead, we want to use a min priority queue based on a heap
- Changes are also needed in the relaxation part of the CRP function to maintain min heap property