

Chapter 5

Network Layer: Control Plane

A note on the use of these PowerPoint slides:

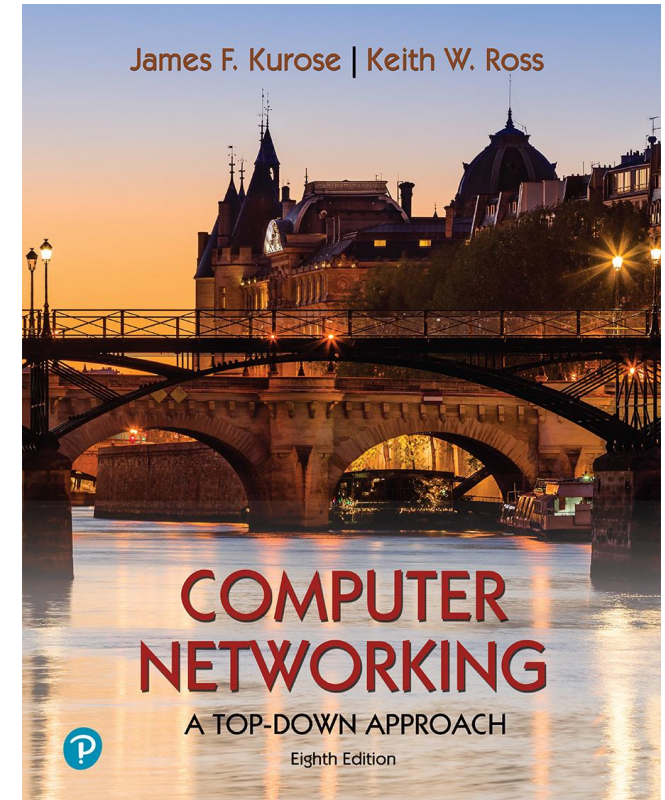
We're making these slides freely available to all (faculty, students, readers). They're in PowerPoint form so you see the animations; and can add, modify, and delete slides (including this one) and slide content to suit your needs. They obviously represent a *lot* of work on our part. In return for use, we only ask the following:

- If you use these slides (e.g., in a class) that you mention their source (after all, we'd like people to use our book!)
- If you post any slides on a www site, that you note that they are adapted from (or perhaps identical to) our slides, and note our copyright of this material.

For a revision history, see the slide note for this page.

Thanks and enjoy! JFK/KWR

All material copyright 1996-2020
J.F Kurose and K.W. Ross, All Rights Reserved



*Computer Networking: A
Top-Down Approach*

8th edition

Jim Kurose, Keith Ross
Pearson, 2020

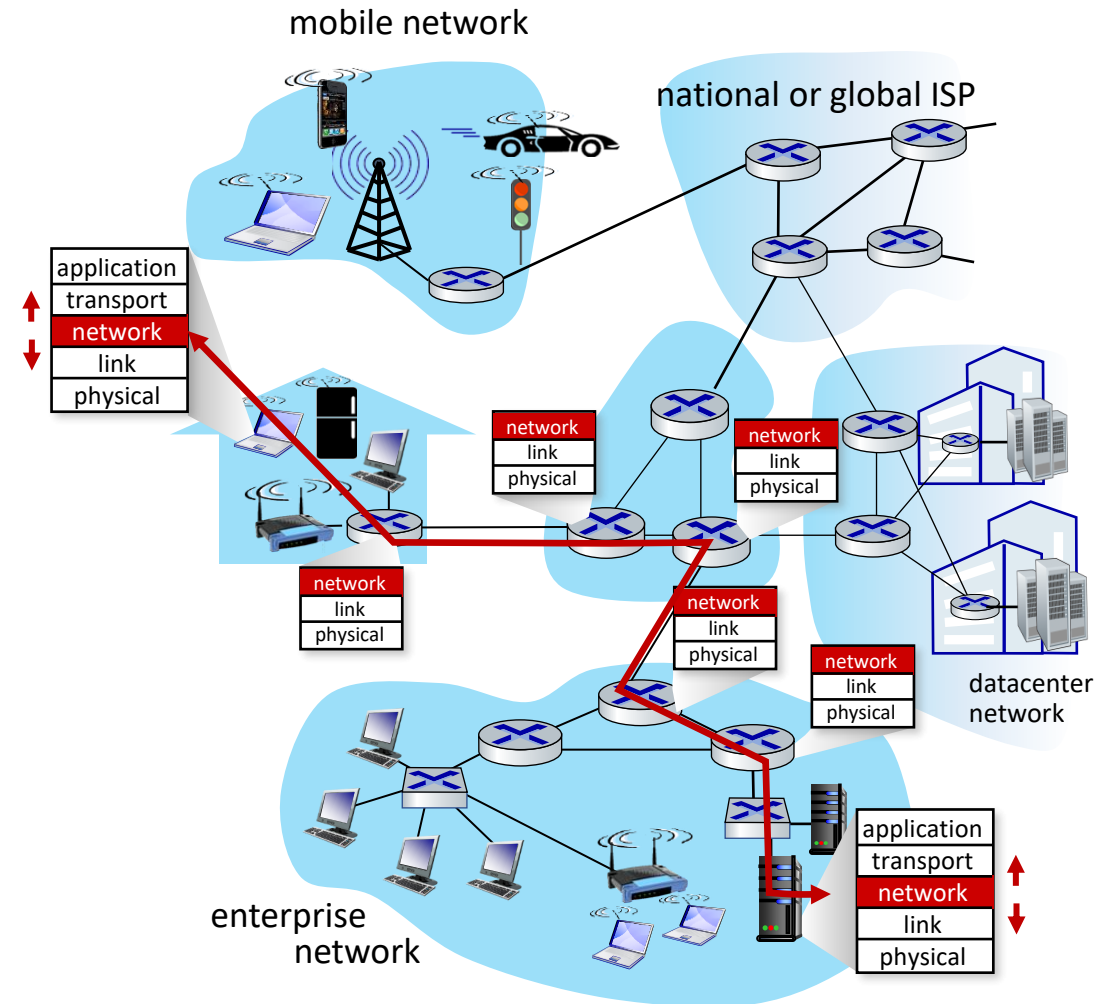
فصل 5

لایه شبکه: صفحه کنترل

الگوریتم های مسیریابی رو میگو اینجا

Network-layer services and protocols

- transport segment from sending to receiving host
 - **sender:** encapsulates segments into datagrams, passes to link layer
 - **receiver:** delivers segments to transport layer protocol
- network layer protocols in *every Internet device*: hosts, routers
- **routers:**
 - examines header fields in all IP datagrams passing through it
 - moves datagrams from input ports to output ports to transfer datagrams along end-end path



در لایه سوم Network-layer قرار دارد و وظیفه اش این است که: انتقال بسته های اطلاعات از مبدا تا مقصد از طریق شبکه

برای اینکه اطلاعات بسته ها از مبدا به مقصد برسند نیازمند یکسری پروتکل هستیم در این لایه که براساس اون پروتکل این کار انجام بشه

این پروتکل های لایه شبکه رو همه اجزای که با لایه شبکه درگیر هستند باید بشناسن و اجرا بکنند البته همه پروتکل ها رو اجرا نمی کنند بلکه اون پروتکل اصلی که پروتکل ip است و پروتکل های مربوط به مسیریابی اینارو همه نود ها باید در طول سطح شبکه اجرا بکنند

یک بسته میاد به روتر و روتر براساس پروتکل ها این رو در مسیر مناسب قرار می ده و این کار برای همه نودها تکرار میشه که به مقصد برسه پس همه نودها درگیر این کار هستند

بسته که این هاست میخواد بفرسته در لایه اپلیکیشن شکل میگیره و می فرسته به لایه transport و لایه transport اینو میده به لایه نت ورک که ارسال بشه به شبکه و ارسال پکت از طریق لایه

لینک و لایه فیزیکی انجام می گیره تا بتونه به نود بعدی برسه و در هر نود این تا لایه نت ورک بالا میاد و کارهایی که لازم است برای پروسس بسته انجام می گیره و دوباره لایه لینک و فیزیکی تا نود بعدی و این ادامه پیدا میکنه تا بسته به مقصد برسه و در مقصد بسته به لایه transport داده میشه و از لایه transport به لایه اپلیکیشن مقصد داده میشه : تو فیلم دو شکلش هم نشون داده می تونی ببینی

Two key network-layer functions

network-layer functions:

- *forwarding*: move packets from a router's input link to appropriate router output link
- *routing*: determine route taken by packets from source to destination
 - *routing algorithms*

analogy: taking a trip

- *forwarding*: process of getting through single interchange
- *routing*: process of planning trip from source to destination



forwarding



routing

دوتا کار اساسی که این پروتکل های مربوط به مسیریابی انجام میدن:

1- forwarding : یک بسته که به یک نود می رسه از پورت ورودی این باید به پورت خروجی ارسال بشه

2- routing : این میگه این بسته باید به کدوم پورت خروجی ارسال بشه پس مشخص کردن این که با توجه به پورت ورودی بسته کدوم پورت خروجی مناسب است توی این بخش انجام میشه توی دنیای واقعی مثلاً می خوایم به یک سفر بریم که از دو بخش تشکیل میشه حرفای بالا:

با توجه به مقصد و مبدا مسافرت از چه مسیری برویم که این همون routing است ینی مسیر رو مشخص میکنیم برای اون مقصد پس طراحی مسیر مناسب برای این مسافرت می شه forwarding اینه که مثلاً رسیدیم اصفهان از مسیر جاده تهران به اصفهان حالا برای اینکه اون مسیری که قبلاً تعیین شده رو بتونیم دنبال بکنیم توی اصفهان باید به کدوم خروجی باید بریم پس توی هر نود توی شبکه هم ما این مسئله رو داریم

Network-layer service model

Network Architecture	Service Model	Quality of Service (QoS) Guarantees ?			
		Bandwidth	Loss	Order	Timing
Internet	best effort	none	no	no	no

Internet “best effort” service model

No guarantees on:

- i. successful datagram delivery to destination
- ii. timing or order of delivery
- iii. bandwidth available to end-end flow

در شبکه best effort ما در مورد این Bandwidth و این ترافیکی که اپلیکیشن داره استفاده میکنه و پکت هایی که قراره بفرسته چقدر Bandwidth استفاده خواهد شد ما هیچ تضمینی درمورد اینکه چقدر Bandwidth در اختیارش خواهد بود نیست

ایا گارانتی میشه که پکت لاس نداشته باشه؟ نه

ایا گارانتی میشه که ترتیب بسته ها حفظ بشه؟ نه

ایا گارانتی میشه در یک زمان بندی مشخصی بسته به مقصد برسه؟ نه

این ویژگی های یک سرویس best effort است که هیچ گارانتی به ما داده نمیشه

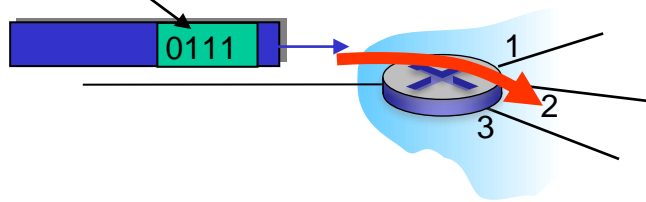
پس بسته رو وقتی که به شبکه میدیم امیدواریم به مقصد برسه و فقط همین و بیشتر از این نمی تونیم از یک شبکه انتظار داشته باشیم

Network layer: data plane, control plane

Data plane:

- *local*, per-router function
- determines how datagram arriving on router input port is forwarded to router output port

values in arriving
packet header



Control plane

- *network-wide* logic
- determines how datagram is routed among routers along end-end path from source host to destination host

حالا این کارها در نودهای توی شبکه که بهشون میگیم روتر یا مسیریاب در Data plane انجام میشه ینی با توجه با دوتا نقشی که هر نود شبکه داره نودهای شبکه را در دوسطح طراحی میکنیم :

Control plane و Data plane

در Data plane همین فورواردینگ انجام میشه ینی بسته ای که می رسه با توجه به ادرس مقصدی که در هدرش است به خروجی مناسب ارسال میشه ولی Control plane این ارتباط رو برقرار میکنه که برای این مقصد کدوم پورت خوبه ینی ارتباط مقصد و پورت رو مشخص میکنه مثالشو نفهمیدم؟؟؟

اون جدولی که پایین نشون دادم رو Control plane اومده برای مقصد های مختلف قبلا مشخص کرده ینی برای هر مقصدی کدوم پورت خروجی مناسب است و اینارو اینجا لیست کرده و ما برای فورواردینگ از اینا استفاده میکنیم

حالا سوال اینه که Control plane این کارو چطوری انجام میده؟ ینی چطوری اطلاعات این جدول پیدا میشه

Out	Port
111	2
?	?

Network-layer functions

- **forwarding**: move packets from router's input to appropriate router output === *data plane*
- **routing**: determine route taken by packets from source to destination === *control plane*
- two control-plane approaches:
 - *traditional routing algorithms*: implemented in routers
 - *software-defined networking (SDN)*: implemented in (remote) servers

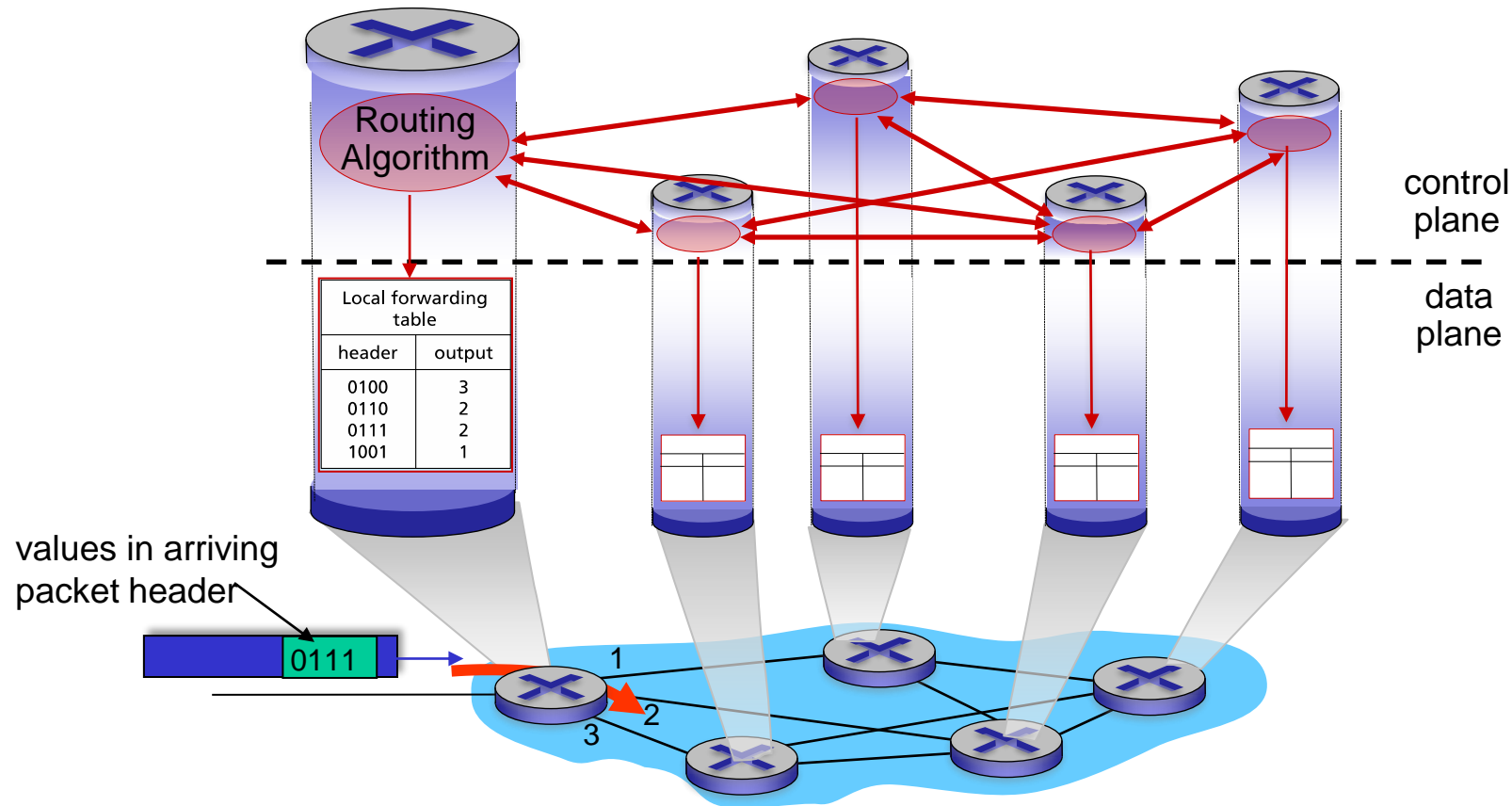
تمام قسمت هایی که درگیر forwarding هستند و خود forwarding توی data plane انجام
میشه

الگوریتم routing توی control plane انجام میشه
پس در یک روتر control plane و data plane داریم

فعلا توی دیدگاه سنتی هستیم ینی هر روتر data plane و control plane توی خودش است

Per-router control plane

Individual routing algorithm components *in each and every router* interact in the control plane

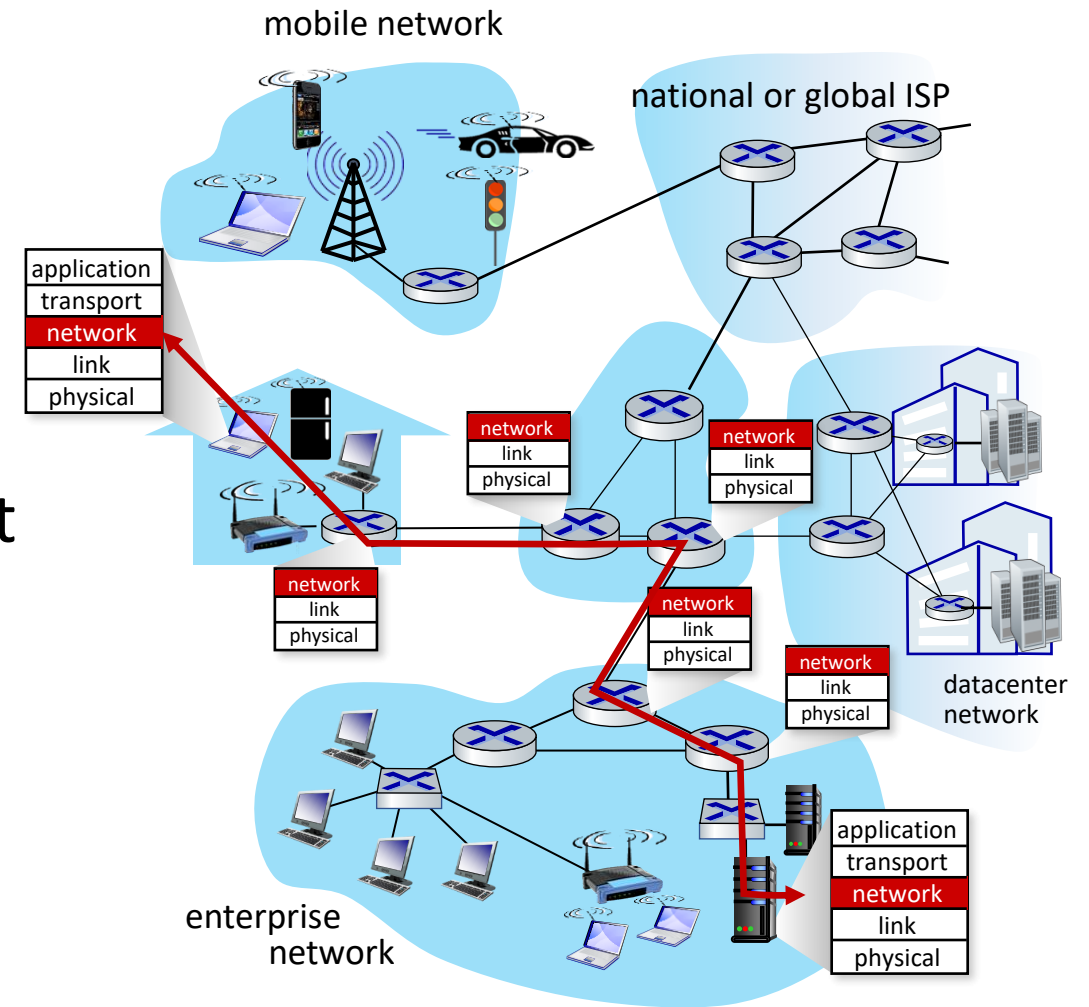


هواییمای کنترلی در هر روتر
اجزای الگوریتم مسیریابی مجزا در هر روتر در صفحه کنترل تعامل دارند

Routing protocols

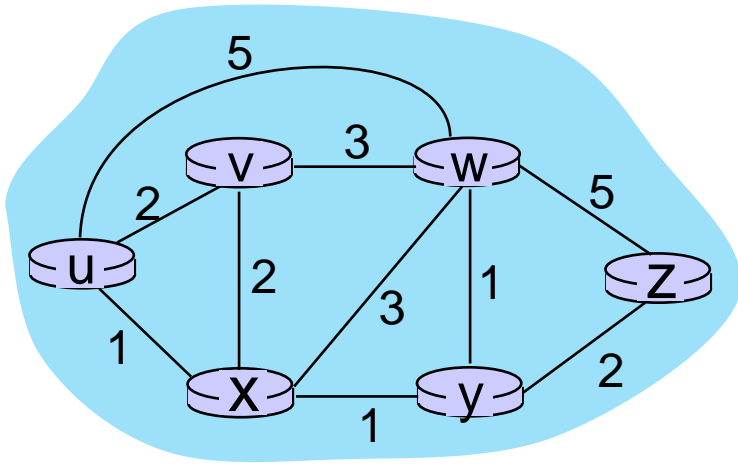
Routing protocol goal: determine “good” paths (equivalently, routes), from sending hosts to receiving host, through network of routers

- **path:** sequence of routers packets traverse from given initial source host to final destination host
- **“good”:** least “cost”, “fastest”, “least congested”
- routing: a “top-10” networking challenge!



پروتکل های Routing مختلفی در شبکه استفاده شده و عمدتاً همه این ها می تونن اطلاعات این جدول رو به دست بیارن و هدف از این پروتکل Routing این است که اولاً مشخص کنه که از برای مقصد مورد نظر چه مسیری وجود داره و کدوم بهترین است پس پروتکل routing هدفش این است که بهترین مسیر را انتخاب بکنه این مسیر که اینجا بهش path یا root میگویم معیارش چیه برای بهترین مسیر بودن؟ به این معیار هزینه مسیر میگویم و ما سعی میکنیم این هزینه رو کم بکنیم حالا این هزینه می تونه هزینه استفاده از لینک باشه یا هزینه پراخت به isp می تونه باشه یا این هزینه می تونه نماینگر پارامترهای دیگری باشه مثلاً سرعت یا تاخیر رسیدن بسته به مقصد و همینطور عرض باند مسیر یا تعداد نودهای میانی ینی مثلاً فرض کنیم تعداد نودهای میانی باعث تاخیر میشن پس تعداد نودهای میانی می تونه معیار خوبی باشه که ما مسیر بهتر را انتخاب بکنیم پس ممکنه معیارهای مختلفی برای بهترین مسیر انتخاب بشه

Graph abstraction: link costs



$c_{a,b}$: cost of *direct* link connecting a and b

e.g., $c_{w,z} = 5$, $c_{u,z} = \infty$

cost defined by network operator:
could always be 1, or inversely related
to bandwidth, or inversely related to
congestion

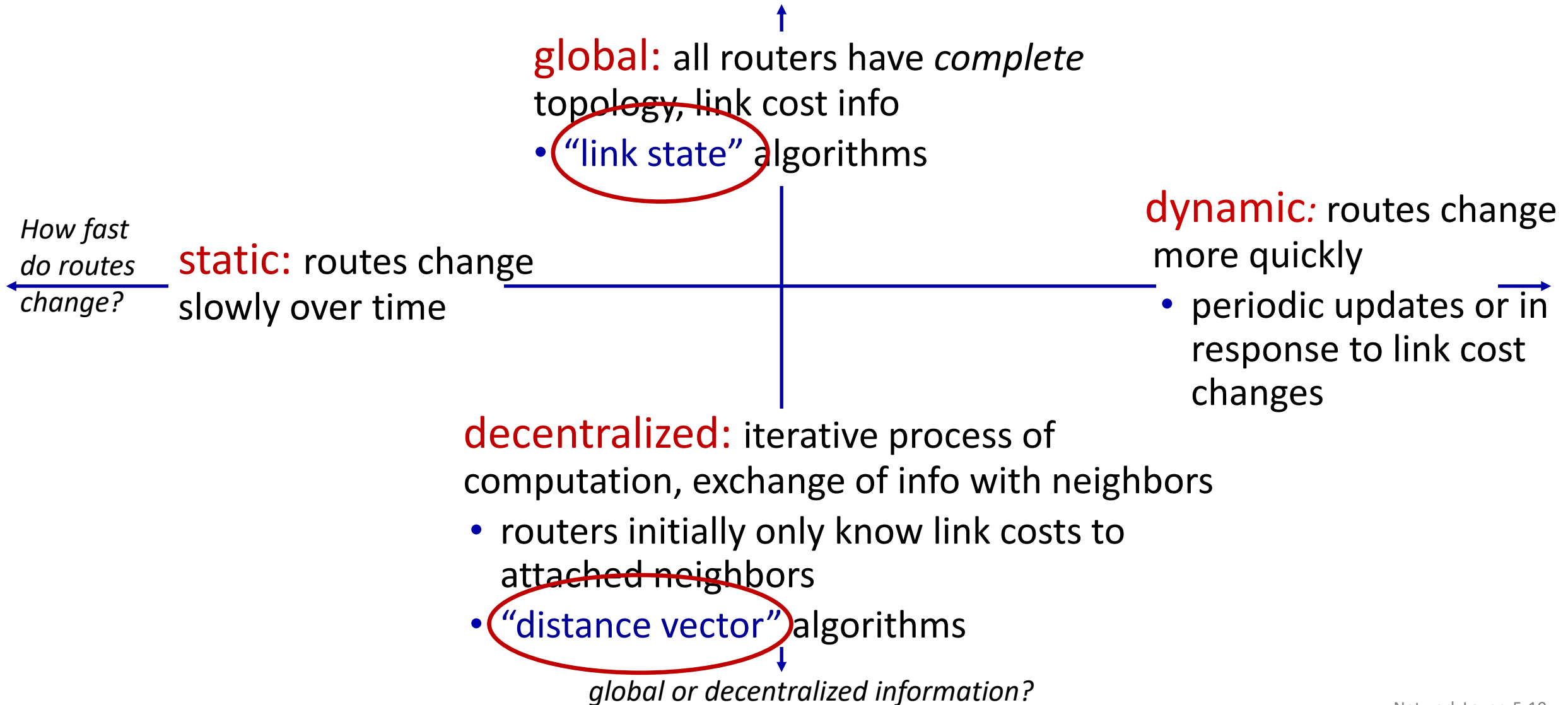
graph: $G = (N, E)$

N : set of routers = $\{ u, v, w, x, y, z \}$

E : set of links = $\{ (u,v), (u,x), (v,x), (v,w), (x,w), (x,y), (w,y), (w,z), (y,z) \}$

پروتکل های مسیریابی عمدتاً مسیر را براساس گراف شبکه به دست میارن
این گراف شبکه از نود و یال تشکیل شده : نودهای این گراف روترهای شبکه هستند و یال های اون
لینک های بین روترهای شبکه
این عددهای روی لینک همون هزینه لینک است که مبنای می تونه چیزهای مختلفی باشه مثلاً
ممکنه طول مسیر باشه یا عرض باند مسیر یا...
بهترین مسیر با توجه به این گراف چه مسیری است؟ بهترین مسیر مسیری است که جمع هزینه های
گام اون مسیر یعنی لینک هایی که بسته ما باید طی بکنه کمتر باشه از همه پس ما هزینه مسیر یا
روت رو براساس جمع لینک های اون مسیر تعریف میکنیم
پس هزینه مسیر یک مقدار جمع شونده است

Routing algorithm classification



الگوریتم های روتینگی که استفاده میکنیم این ها انواع مختلفی دارند:

برخی از این الگوریتم های مسیریابی اطلاعاتشون گلوبال است یعنی همه اطلاعات توپولوژی شبکه دارند یعنی همه نود ها اطلاعات این گراف رو دارند (الگوریتم های **link state** الگوریتم هایی هستند که همه اطلاعات نودها بهم دیگه منتقل میکنن تا در نهایت همه، اطلاعات کل شبکه رو داشته باشند)

الگوریتم **decentralized** : هر نود فقط اطلاعات خودشو داره و همسایه هاشو از بقیه شبکه اطلاعاتی نداره یعنی هر نود بخشی از اطلاعات رو داره - هر نود براساس اینکه همسایه هاش کی هستند و فاصله خودش تا همسایه ها و اینکه همسایه ها چه مسیرهایی رو برای مقصد توی شبکه دارند مقصد خودش واسه شبکه رو پیدا میکنه

الگوریتم های **static** الگوریتم هایی هستند که به طور ثابت مسیر را مشخص میکنند و این می مونه مثل تابلوهای راهنمایی و رانندگی توی شهر

الگوریتم های **dynamic** اونایی هست که براساس وضعیت شبکه تغییر میکنه یعنی الگوریتم ما براساس وضعیت شبکه ممکنه مسیرها رو تغییر بده

Dijkstra's link-state routing algorithm

- **centralized:** network topology, link costs known to *all* nodes
 - accomplished via “link state broadcast”
 - all nodes have same info
- computes least cost paths from one node (“source”) to all other nodes
 - gives *forwarding table* for that node
- **iterative:** after k iterations, know least cost path to k destinations

notation

- $c_{x,y}$: direct link cost from node x to y ; $= \infty$ if not direct neighbors
- $D(v)$: *current* estimate of cost of least-cost-path from source to destination v
- $p(v)$: predecessor node along path from source to v
- N' : set of nodes whose least-cost-path *definitively* known

الگوریتم های link-state معروف ترینش Dijkstra است و در الگوریتم Dijkstra ما فرض میکنیم که هر نود اطلاعات شبکه رو داره و این اطلاعات از طریق شبکه توسط روترها منتقل میشه ینی هر روتر لینک هایی که بین خودشو همسایه ها است رو لیست میکنه و هزینه رو هم لیست میکنه و اینو برای همه روترهای شبکه می فرسته پس یک روتر لینک های خودش رو به همسایه هاش می فرسته و روتر های دیگر هم همین کارو میکنن پس هر روتر این بردکست هارو از بقیه روترها دریافت میکنه و اطلاعات کل شبکه رو بازسازی میکنه

بعد الگوریتم رو همون گراف اجرا میکنه و اون نود خودش رو به عنوان مبدا فرض میکنه و همه مقصد های که توی شبکه قابل تصور است که در واقع نودهای دیگر همون گراف هستنند به عنوان مقصد بهترین مسیر را پیدا میکنه پس یک درخت به دست میاد که ریشش همون نود است و شاخه هاش ینی برگ هاش مقصد های نودهای دیگر شبکه است و براساس این درخت اطلاعات forwarding table به دست میاد

Dijkstra's link-state routing algorithm

1 *Initialization:*

2 $N' = \{u\}$ /* compute least cost path from u to all other nodes */

3 for all nodes v

4 if v adjacent to u /* u initially knows direct-path-cost only to direct neighbors */

5 then $D(v) = c_{u,v}$ /* but may not be *minimum* cost! */

6 else $D(v) = \infty$

7



8 *Loop*

9 find w not in N' such that $D(w)$ is a minimum

10 add w to N'

11 update $D(v)$ for all v adjacent to w and not in N' :

12 **$D(v) = \min (D(v), D(w) + c_{w,v})$**

13 /* new least-path-cost to v is either old least-cost-path to v or known

14 least-cost-path to w plus direct-cost from w to v */

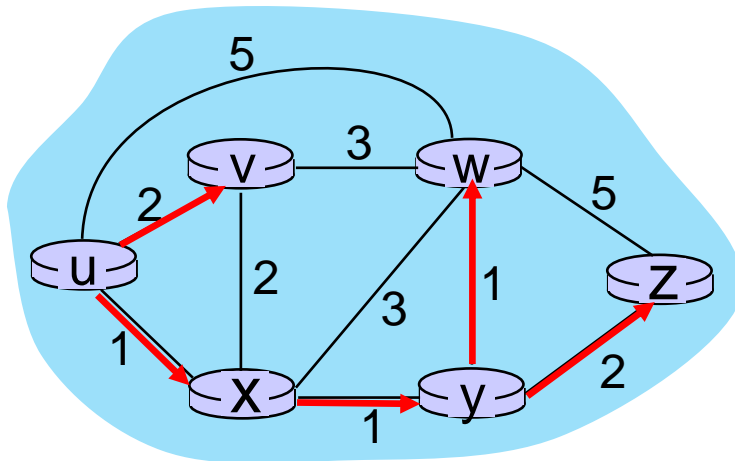
15 *until all nodes in N'*

n اپرین در ابتدا فقط u ینی همون نود اولیه که انتخاب کردیم داخلش هست و بقیه نودهای گرافمون v است و نودهای دیگر اگر همسایه مستقیم u هستند ینی لینک مستقیم بینشون است میشه $D(v)$ و اگر همسایه مستقیم نباشه می شه بی نهایت و وارد حلقه میشیم :

در این حلقه از بین هزینه های بقیه نودها ینی فاصله اش و بعد این فاصله ها رو مقایسه میکنیم و اونی که کمترین فاصله رو داره به n اپرین اضافه میکنیم و بعد $D(v)$ را اپدیت میکنیم در نهایت همه نودها میان توی n اپرین و در این حالت الگوریتم به پایان می رسه

Dijkstra's algorithm: an example

Step	N'	$D(v), p(v)$	$D(w), p(w)$	$D(x), p(x)$	$D(y), p(y)$	$D(z), p(z)$
0	u	2, u	5, u	1, u	∞	∞
1	ux	2, u	4, x		2, x	∞
2	uxy	2, u	3, y			4, y
3	uxyv		3, y			4, y
4	uxyvw					4, y
5	uxyvwz					



Initialization (step 0): For all a : if a adjacent to then $D(a) = c_{u,a}$

find a not in N' such that $D(a)$ is a minimum

add a to N'

update $D(b)$ for all b adjacent to a and not in N' :

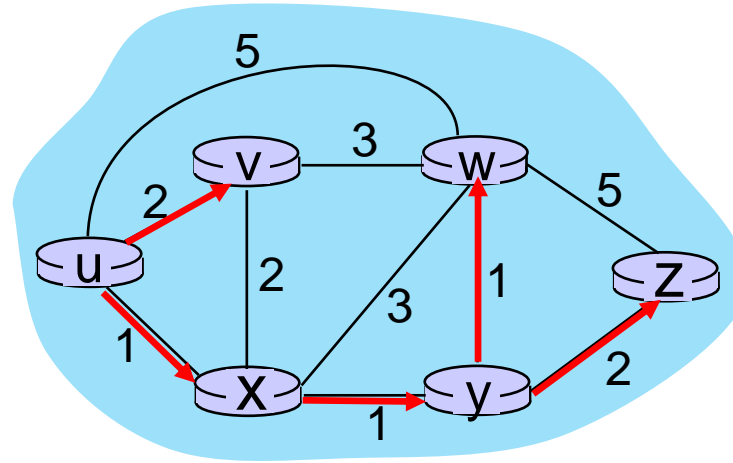
$$D(b) = \min (D(b), D(a) + c_{a,b})$$

این یک درخت پوشا است

-

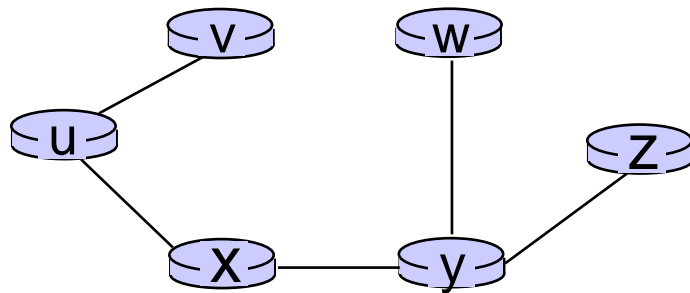
از u می‌خواهیم شروع بکنیم:
 n اپرین لیست نودهایی که مسیرشون مشخص شده

Dijkstra's algorithm: an example



این جدولی است که برای forwarding آماده میشه

resulting least-cost-path tree from u:



resulting forwarding table in u:

destination	outgoing link
v	(u,v)
x	(u,x)
y	(u,x)
w	(u,x)
z	(u,x)

route from u to v directly

route from u to all other destinations via x

مثلا توی جدول forwarding می گه برای مقصد v باید به لینک (u,v) ارسال بشه و برای بقیه هم نوشته توی جدول

پس این اون جدولی است که توی روتر توی دیتاپلن نوشته میشه و دیتاپلن از این استفاده میکنه برای forwarding

به این جدول میگن forwarding label یا routing label

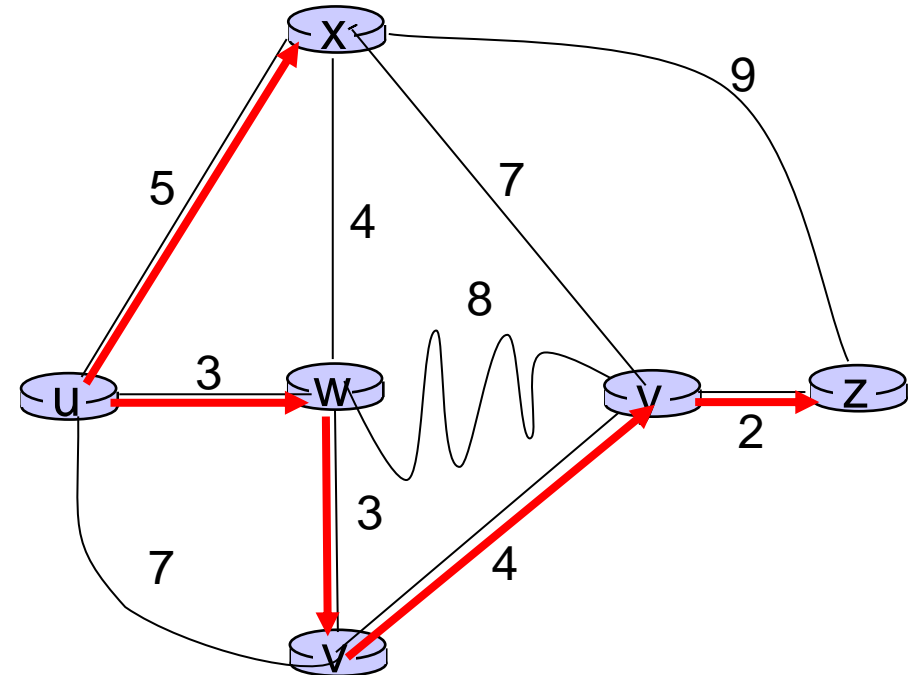
این جدولی است که برای نود u به دست آوردیم و همینطور بقیه نودها هم باید همین کارو بکنند تا بتونند جدولشون رو به دست بیاورند مثلا x هم همین کارو انجام میده تا جدولش به دست بیاد و..

حالا اگر بسته ای برسه به u تا به مقصد z برسه بسته توی u طبق جدولش ارسال میشه به x و بعد طبق جدول x می فرسته به y و بعد طبق جدول y می فرسته به z و تمام پس هر نودی یک جدول داره که اون جدول رو به دست آورده

نکته: پکت میتونه به هر نودی وارد بشه حتما اینطوری نیست که همیشه نود u مبدا باشه چون اینجا پکت به نود u وارد شده u شده مبدا

Dijkstra's algorithm: another example

Step	N'	$D(v), p(v)$	$D(w), p(w)$	$D(x), p(x)$	$D(y), p(y)$	$D(z), p(z)$
0	u	7, u	3, u	5, u	∞	∞
1	uw	6, w		5, u	11, w	∞
2	uwX	6, w			11, w	14, x
3	uwXv				10, v	14, x
4	uwXvy					12, y
5	uwXvyz					



notes:

- construct least-cost-path tree by tracing predecessor nodes
- ties can exist (can be broken arbitrarily)

Dijkstra's algorithm: discussion

algorithm complexity: n nodes

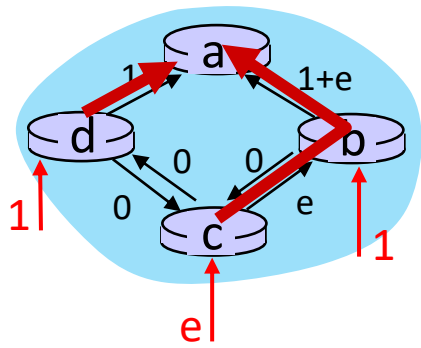
- each of n iteration: need to check all nodes, w , not in N
- $n(n+1)/2$ comparisons: $O(n^2)$ complexity
- more efficient implementations possible: $O(n \log n)$

message complexity:

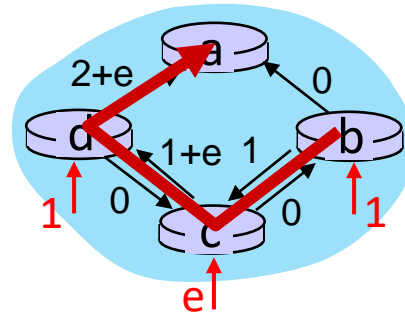
- each router must *broadcast* its link state information to other n routers
- efficient (and interesting!) broadcast algorithms: $O(n)$ link crossings to disseminate a broadcast message from one source
- each router's message crosses $O(n)$ links: overall message complexity: $O(n^2)$

Dijkstra's algorithm: oscillations possible

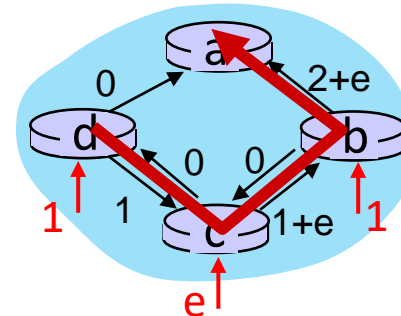
- when link costs depend on traffic volume, **route oscillations** possible
- sample scenario:
 - routing to destination a, traffic entering at d, c, e with rates 1, e (<1), 1
 - link costs are directional, and volume-dependent



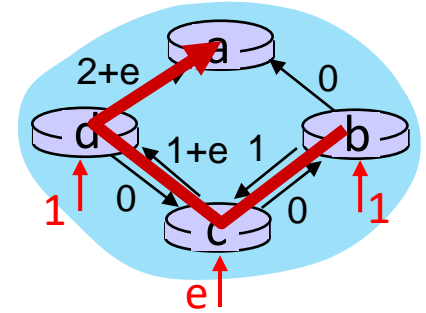
initially



given these costs,
find new routing....
resulting in new costs



given these costs,
find new routing....
resulting in new costs



given these costs,
find new routing....
resulting in new costs

Distance vector algorithm

Based on *Bellman-Ford* (BF) equation (dynamic programming):

Bellman-Ford equation

Let $D_x(y)$: cost of least-cost path from x to y .

Then:

$$D_x(y) = \min_v \{ c_{x,v} + D_v(y) \}$$

v 's estimated least-cost-path cost to y

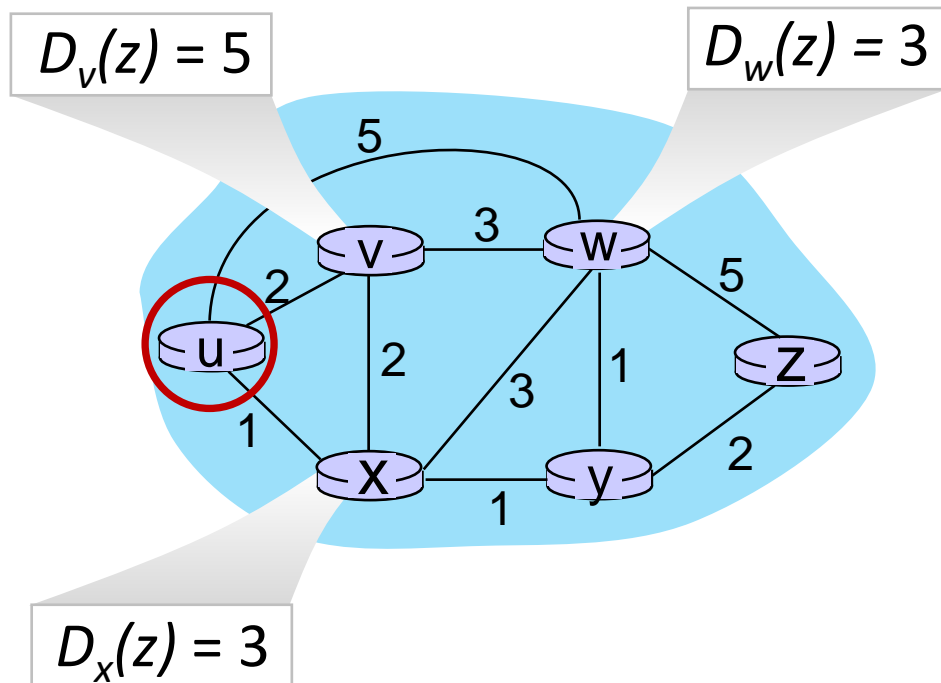
\min taken over all neighbors v of x

direct cost of link from x to v

نوع دیگر الگوریتم هایی که توی شبکه استفاده میشه الگوریتم های Distance vector پروتکل Distance vector الگوریتم مشخصی که توشون استفاده میشه الگوریتم Bellman-Ford است و در الگوریتم Bellman-Ford اگر فرض کنیم بهترین مسیر با کوتاه ترین هزینه از مبدا x است تا مقصد y (ینی $D(x,y)$) در این حالت این مسیر از این نود برای اون مقصد لاجرم از طریق یکی از همسایه های این نود باید عبور بکنه پس کافیه اینجا بگیریم از طریق کدوم همسایه ما بهترین مسیر رو داریم این همسایه ها رو اگر با مجموعه V نشون بدیم و بعد روی V مینیمم رو پیدا میکنیم روی چی؟ روی هزینه ای که از طریق اون همسایه داره C هزینه است که از نود x به نود v است $D(v,y)$: هزینه اون همسایه تا اون مقصد است مثلاً اگر سه تا همسایه داریم هزینه خودم تا اون همسایه چقدره ینی همون C و بعد همسایه تا اون مقصد چقدره ینی D $D(v,y)$ را ما محاسبه نمیکنیم فرض میکنیم اون همسایه خودش محاسبه کرده اینو این کار بالا رو که گفتیم برای همه مقصد های شبکه به دست میاریم و در نهایت اون درخته رو به دست میاریم و بعد براساس اون جدول را به دست میاریم

Bellman-Ford Example

Suppose that u 's neighboring nodes, x, v, w , know that for destination z :



Bellman-Ford equation says:

$$\begin{aligned} D_u(z) &= \min \{ c_{u,v} + D_v(z), \\ &\quad c_{u,x} + D_x(z), \\ &\quad c_{u,w} + D_w(z) \} \\ &= \min \{ 2 + 5, \\ &\quad 1 + 3, \\ &\quad 5 + 3 \} = 4 \end{aligned}$$

node achieving minimum (x) is next hop on estimated least-cost path to destination (z)

فرض می کنیم توی این گراف همسایه های U به عنوان مبدا مورد نظر ما همسایه هاش X, V, W است و فرض میکنیم هزینه هاشون تا مقصد Z برای این همسایه ها رو داریم ینی فرض میکنیم همسایه ها بهترین مسیر را قبلا برای مقصد Z به دست آوردن ینی از خودشون تا مقصد Z مثلا $D_V(Z)$ اینجا 5 است و...

و بعد بقیه مراحل رو از توی اسلاید ببین...

همین کارو برای هر مقصد دیگری به جز Z هم انجام میدیم تا جدول U برای همه مقصدهاش به دست بیاد

Distance vector algorithm

key idea:

- from time-to-time, each node sends its own distance vector estimate to neighbors
- when x receives new DV estimate from any neighbor, it updates its own DV using B-F equation:

$$D_x(y) \leftarrow \min_v \{c_{x,v} + D_v(y)\} \text{ for each node } y \in N$$

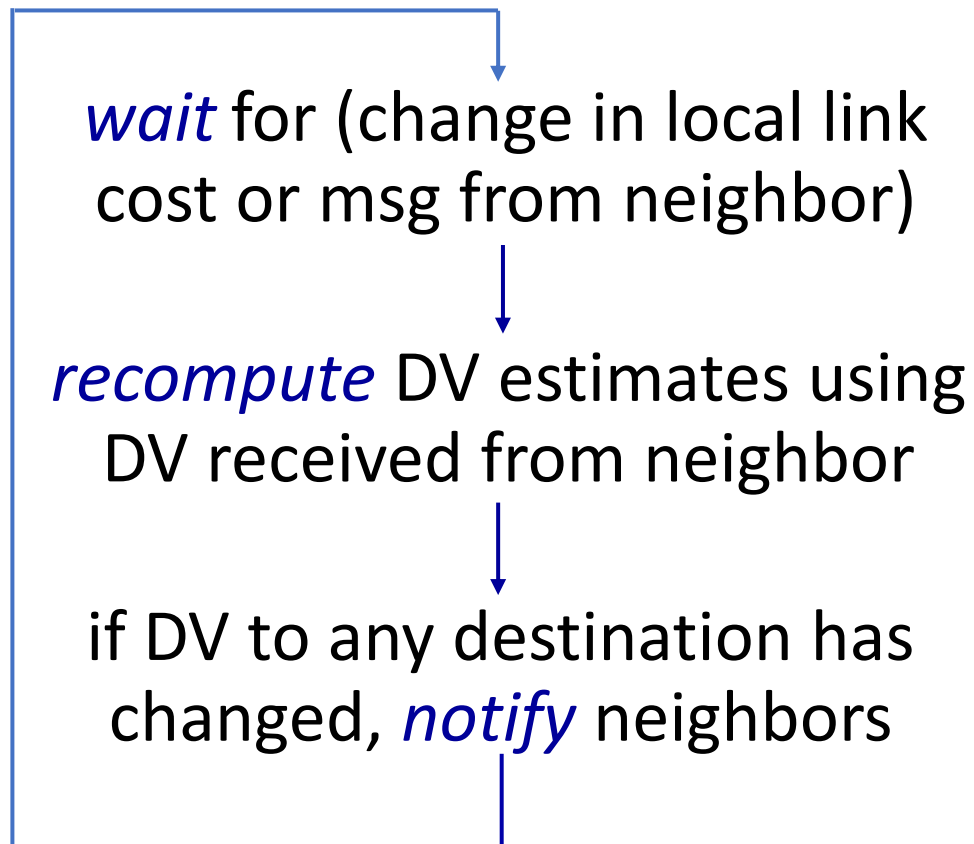
- under minor, natural conditions, the estimate $D_x(y)$ converge to the actual least cost $d_x(y)$

پس الگوریتم Distance vector به این ترتیب عمل میکند فرض میکند که $D_x(y)$ برای همه مقصد ها از x به دست آمده و x اینارو داره که به این میگیریم Distance vector ینی هر نود Distance vector خودشو برای همه مقصدها داره

نود دلخواه x با توجه به اطلاعاتی که از همسایه هاش داره و با توجه به Distance vector همسایه هاش میاد Distance vector خودشو برای همه مقصدها آپدیت میکند و برای این لازمه Distance vector همه همسایه هاشو داشته باشه
الگوریتم اینجوری عمل میکنه:

Distance vector algorithm:

each node:



iterative, asynchronous: each local iteration caused by:

- local link cost change
- DV update message from neighbor

distributed, self-stopping: each node notifies neighbors *only* when its DV changes

- neighbors then notify their neighbors – *only if necessary*
- no notification received, no actions taken!

نکته: اگر تغییری توی شبکه اتفاق بیوفته یک لینکی قطع بشه یا down بشه یا یه نودی down بشه الگوریتم دوباره باید اجرا بشه

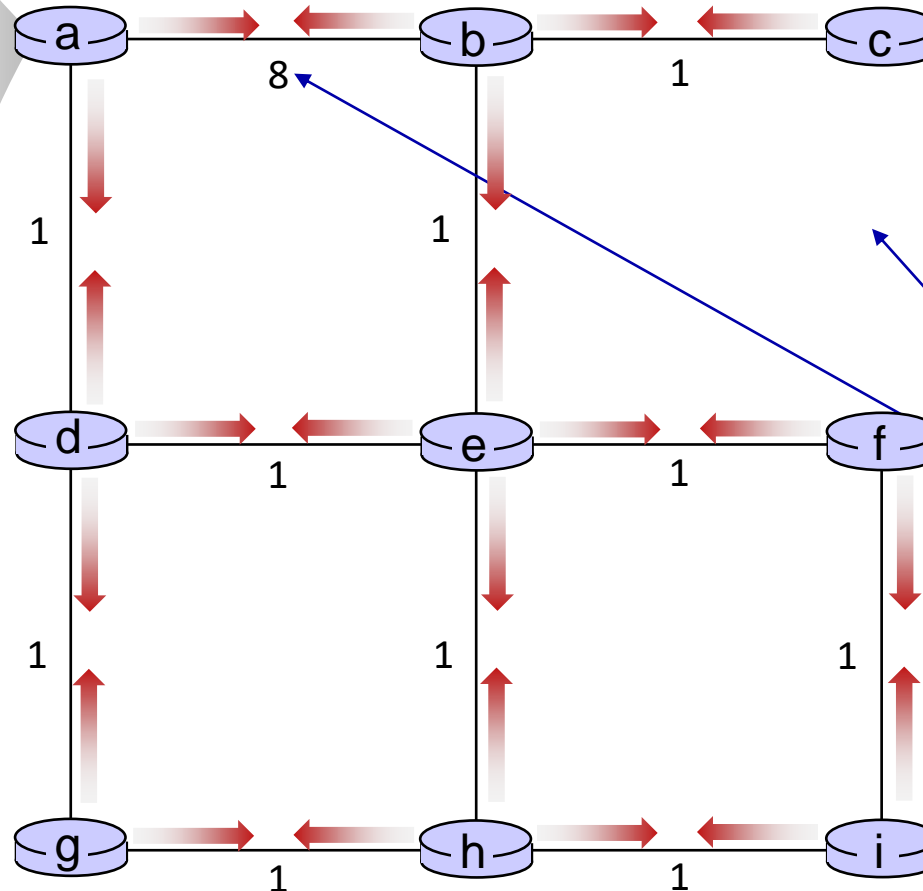
Distance vector: example



t=0

- All nodes have distance estimates to nearest neighbors (only)
- All nodes send their local distance vector to their neighbors

DV in a:
$D_a(a)=0$
$D_a(b)=8$
$D_a(c)=\infty$
$D_a(d)=1$
$D_a(e)=\infty$
$D_a(f)=\infty$
$D_a(g)=\infty$
$D_a(h)=\infty$
$D_a(i)=\infty$



A few asymmetries:
■ missing link
■ larger cost

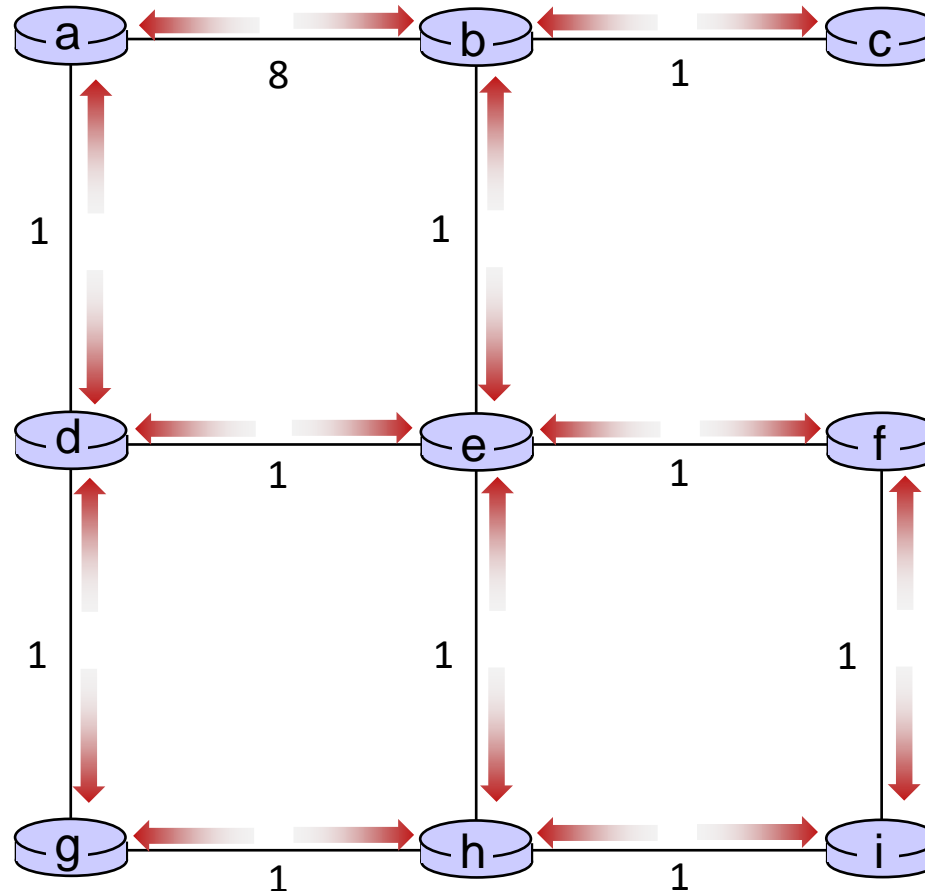
Distance vector example: iteration



$t=1$

All nodes:

- receive distance vectors from neighbors
- compute their new local distance vector
- send their new local distance vector to neighbors



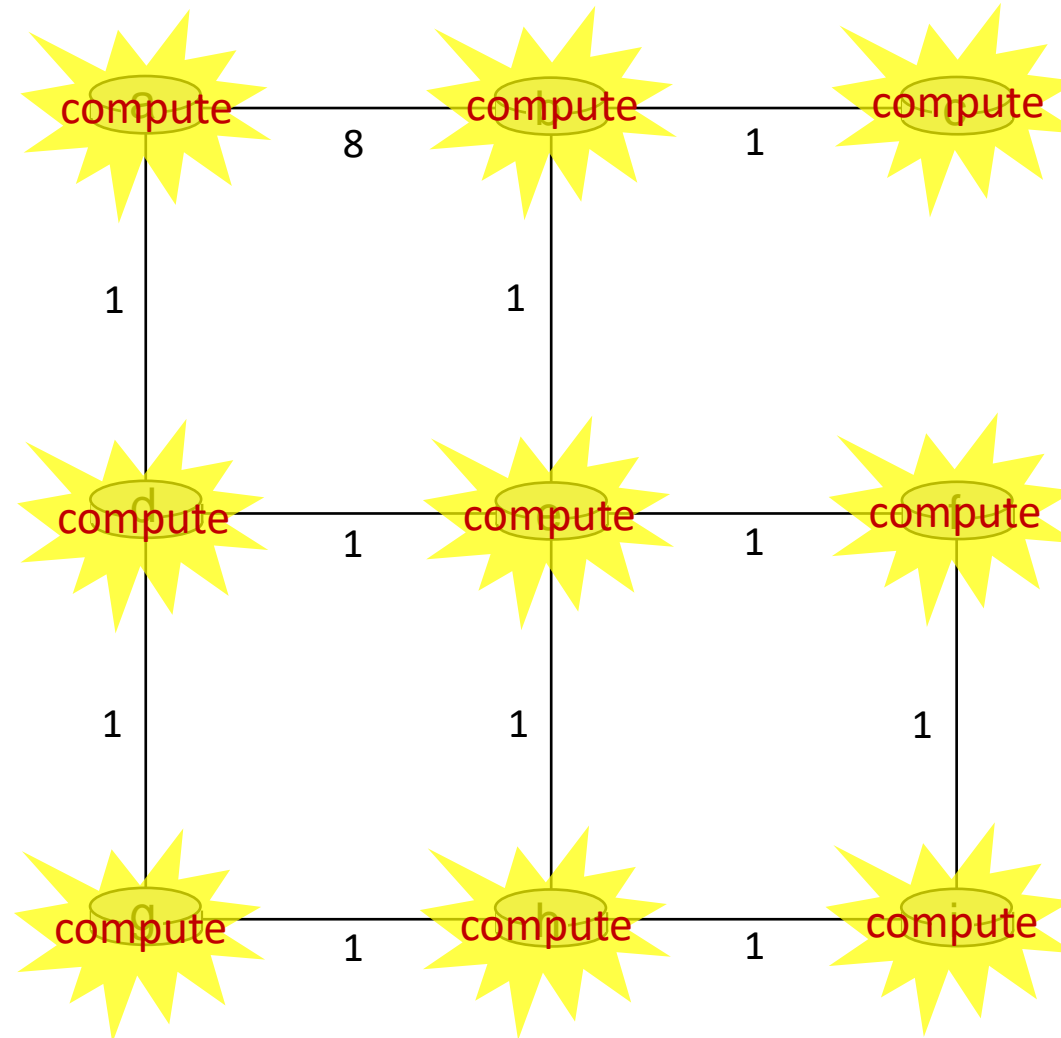
Distance vector example: iteration



$t=1$

All nodes:

- receive distance vectors from neighbors
- compute their new local distance vector
- send their new local distance vector to neighbors



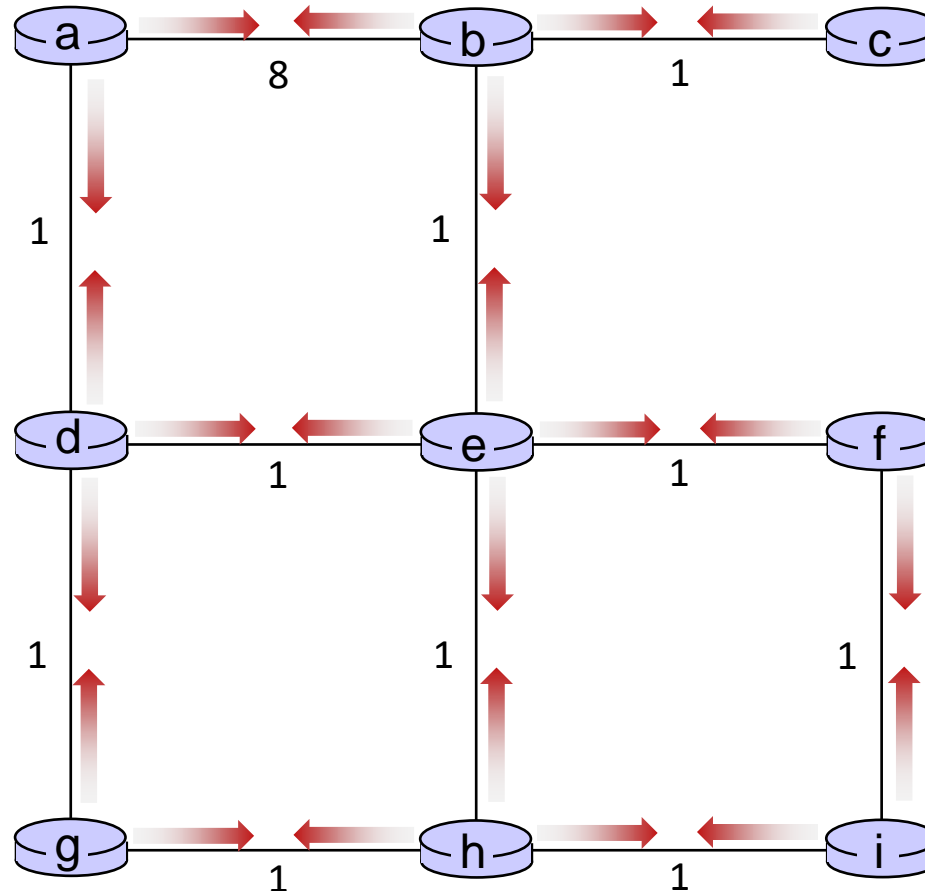
Distance vector example: iteration



$t=1$

All nodes:

- receive distance vectors from neighbors
- compute their new local distance vector
- send their new local distance vector to neighbors



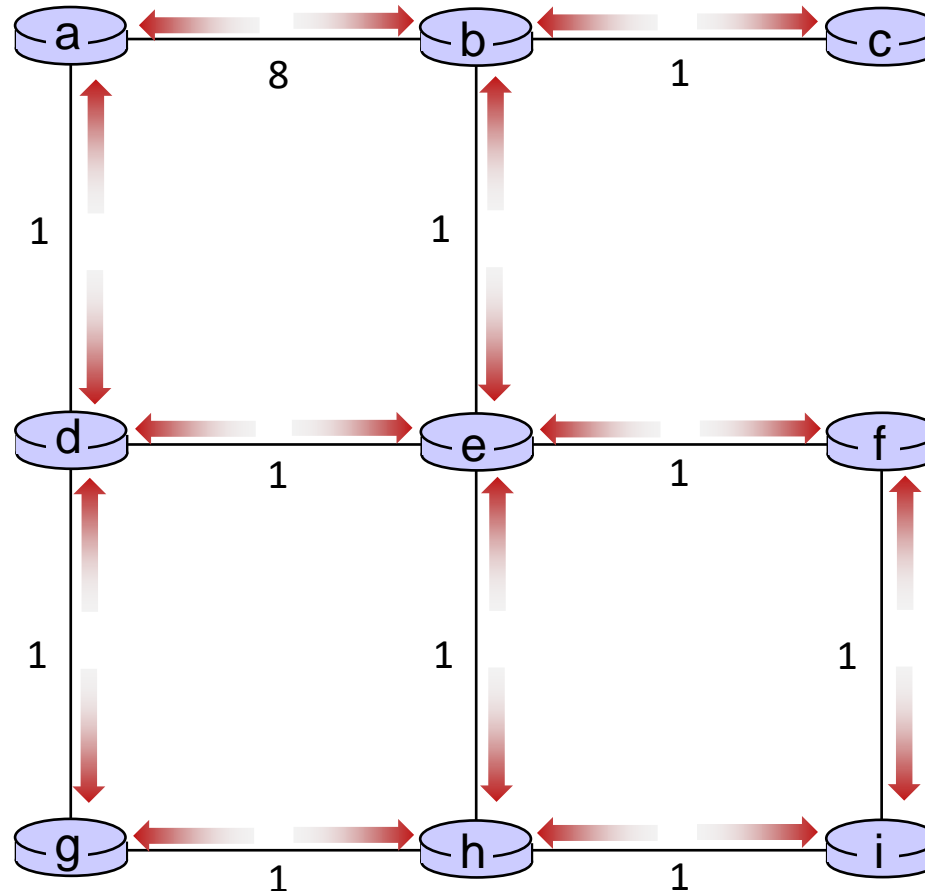
Distance vector example: iteration



t=2

All nodes:

- receive distance vectors from neighbors
- compute their new local distance vector
- send their new local distance vector to neighbors



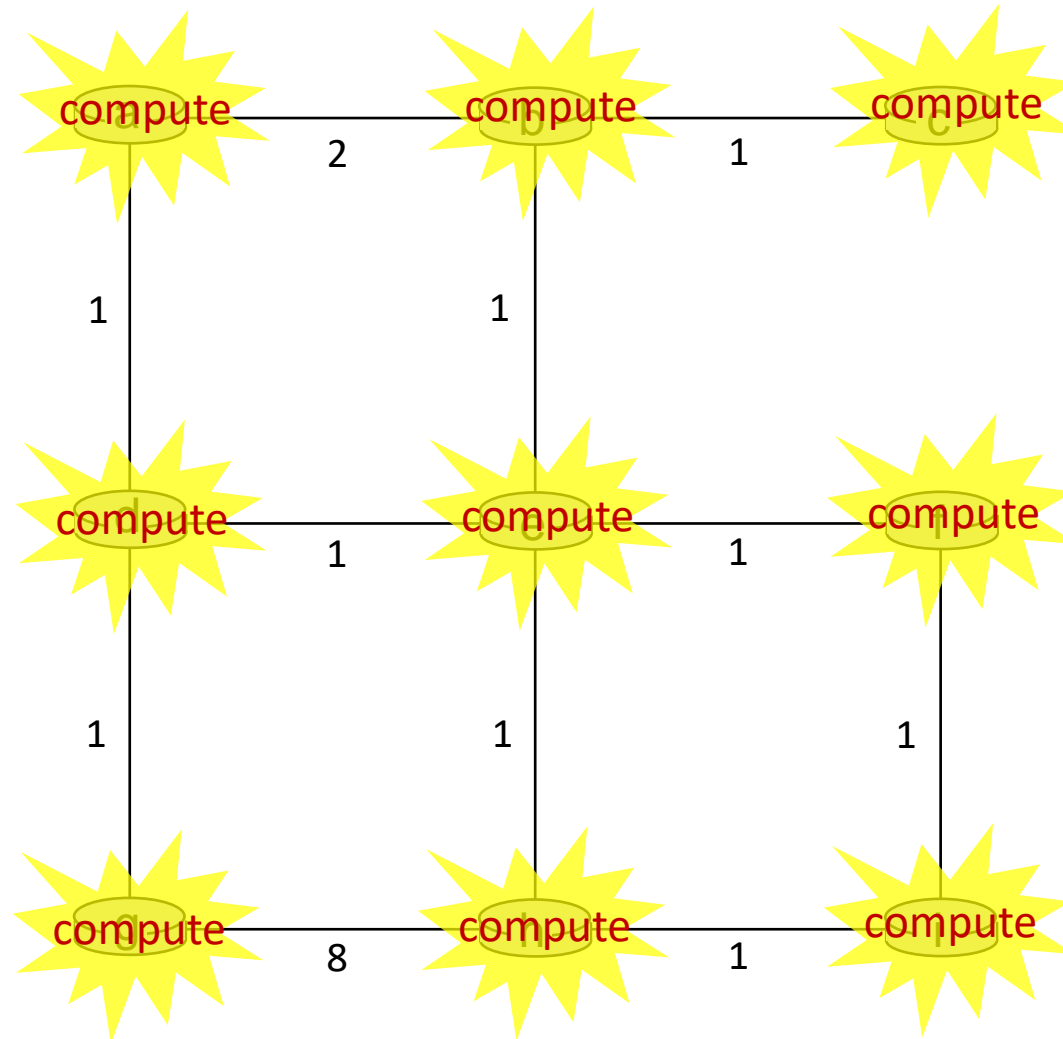
Distance vector example: iteration



t=2

All nodes:

- receive distance vectors from neighbors
- compute their new local distance vector
- send their new local distance vector to neighbors



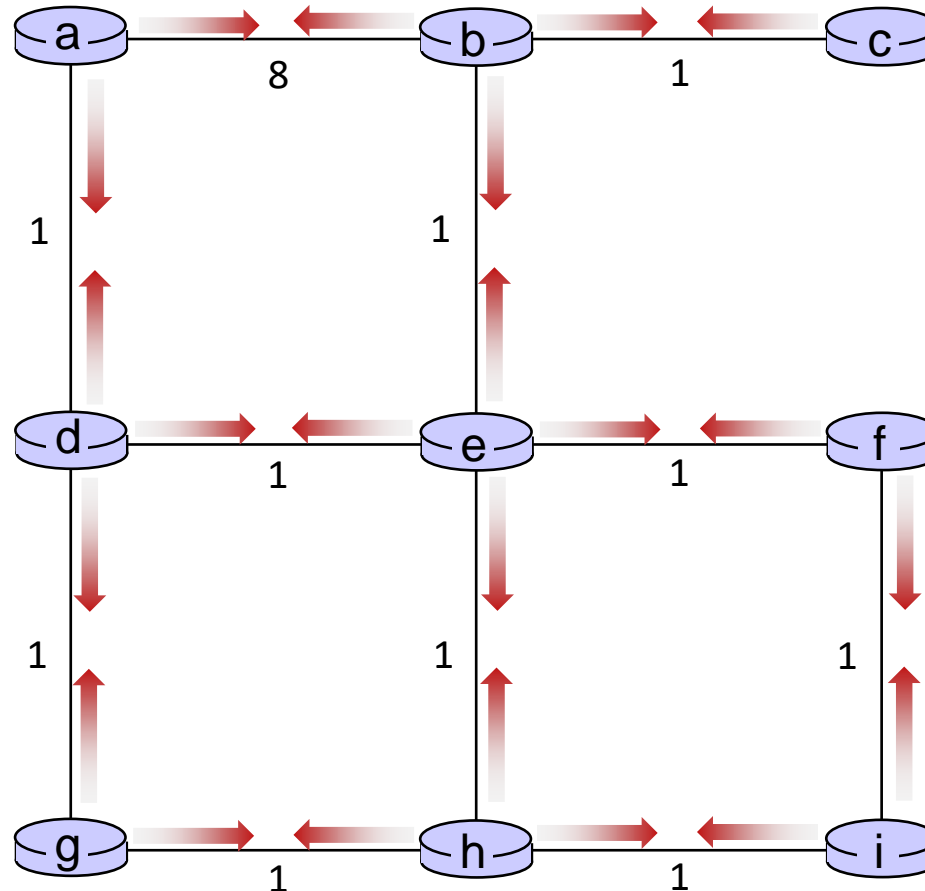
Distance vector example: iteration



t=2

All nodes:

- receive distance vectors from neighbors
- compute their new local distance vector
- send their new local distance vector to neighbors



Distance vector example: iteration

.... and so on

Let's next take a look at the iterative *computations* at nodes

Distance vector example:



t=1

- b receives DVs from a, c, e

DV in a:
$D_a(a) = 0$
$D_a(b) = 8$
$D_a(c) = \infty$
$D_a(d) = 1$
$D_a(e) = \infty$
$D_a(f) = \infty$
$D_a(g) = \infty$
$D_a(h) = \infty$
$D_a(i) = \infty$

DV in b:

$$D_b(a) = 8$$

$$D_b(c) = 1$$

$$D_b(d) = \infty$$

$$D_b(e) = 1$$

$$D_b(f) = \infty$$

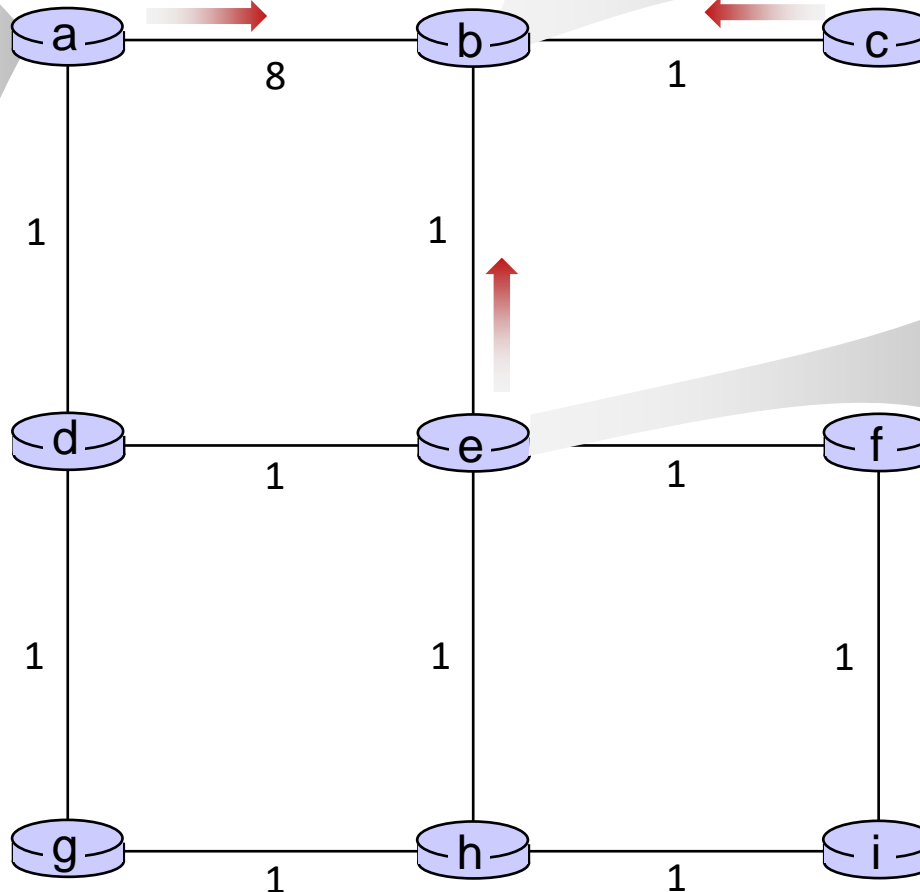
$$D_b(g) = \infty$$

$$D_b(h) = \infty$$

$$D_b(i) = \infty$$

DV in c:
$D_c(a) = \infty$
$D_c(b) = 1$
$D_c(c) = 0$
$D_c(d) = \infty$
$D_c(e) = \infty$
$D_c(f) = \infty$
$D_c(g) = \infty$
$D_c(h) = \infty$
$D_c(i) = \infty$

DV in e:
$D_e(a) = \infty$
$D_e(b) = 1$
$D_e(c) = \infty$
$D_e(d) = 1$
$D_e(e) = 0$
$D_e(f) = 1$
$D_e(g) = \infty$
$D_e(h) = 1$
$D_e(i) = \infty$



Distance vector example:

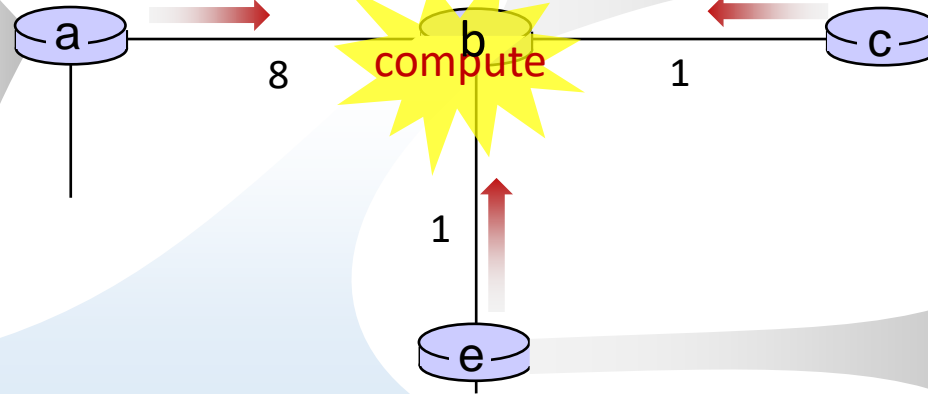


t=1

- b receives DVs from a, c, e, computes:

$$\begin{aligned}
 D_b(a) &= \min\{c_{b,a} + D_a(a), c_{b,c} + D_c(a), c_{b,e} + D_e(a)\} = \min\{8, \infty, \infty\} = 8 \\
 D_b(c) &= \min\{c_{b,a} + D_a(c), c_{b,c} + D_c(c), c_{b,e} + D_e(c)\} = \min\{\infty, 1, \infty\} = 1 \\
 D_b(d) &= \min\{c_{b,a} + D_a(d), c_{b,c} + D_c(d), c_{b,e} + D_e(d)\} = \min\{9, 2, \infty\} = 2 \\
 D_b(e) &= \min\{c_{b,a} + D_a(e), c_{b,c} + D_c(e), c_{b,e} + D_e(e)\} = \min\{\infty, \infty, 1\} = 1 \\
 D_b(f) &= \min\{c_{b,a} + D_a(f), c_{b,c} + D_c(f), c_{b,e} + D_e(f)\} = \min\{\infty, \infty, 2\} = 2 \\
 D_b(g) &= \min\{c_{b,a} + D_a(g), c_{b,c} + D_c(g), c_{b,e} + D_e(g)\} = \min\{\infty, \infty, \infty\} = \infty \\
 D_b(h) &= \min\{c_{b,a} + D_a(h), c_{b,c} + D_c(h), c_{b,e} + D_e(h)\} = \min\{\infty, \infty, 2\} = 2 \\
 D_b(i) &= \min\{c_{b,a} + D_a(i), c_{b,c} + D_c(i), c_{b,e} + D_e(i)\} = \min\{\infty, \infty, \infty\} = \infty
 \end{aligned}$$

DV in a:
$D_a(a) = 0$
$D_a(b) = 8$
$D_a(c) = \infty$
$D_a(d) = 1$
$D_a(e) = \infty$
$D_a(f) = \infty$
$D_a(g) = \infty$
$D_a(h) = \infty$
$D_a(i) = \infty$



DV in b:

$$D_b(a) = 8$$

$$D_b(c) = 1$$

$$D_b(d) = \infty$$

$$D_b(e) = 1$$

$$D_b(f) = \infty$$

$$D_b(g) = \infty$$

$$D_b(h) = \infty$$

$$D_b(i) = \infty$$

DV in c:
$D_c(a) = \infty$
$D_c(b) = 1$
$D_c(c) = 0$
$D_c(d) = \infty$
$D_c(e) = \infty$
$D_c(f) = \infty$
$D_c(g) = \infty$
$D_c(h) = \infty$
$D_c(i) = \infty$

DV in e:
$D_e(a) = \infty$
$D_e(b) = 1$
$D_e(c) = \infty$
$D_e(d) = 1$
$D_e(e) = 0$
$D_e(f) = 1$
$D_e(g) = \infty$
$D_e(h) = 1$
$D_e(i) = \infty$

DV in b:

$D_b(a) = 8$	$D_b(f) = 2$
$D_b(c) = 1$	$D_b(g) = \infty$
$D_b(d) = 2$	$D_b(h) = 2$
$D_b(e) = 1$	$D_b(i) = \infty$

Distance vector example:



t=1

- c receives DVs from b

DV in a:

$D_a(a)=0$
 $D_a(b)=8$
 $D_a(c)=\infty$
 $D_a(d)=1$
 $D_a(e)=\infty$
 $D_a(f)=\infty$
 $D_a(g)=\infty$
 $D_a(h)=\infty$
 $D_a(i)=\infty$

DV in b:

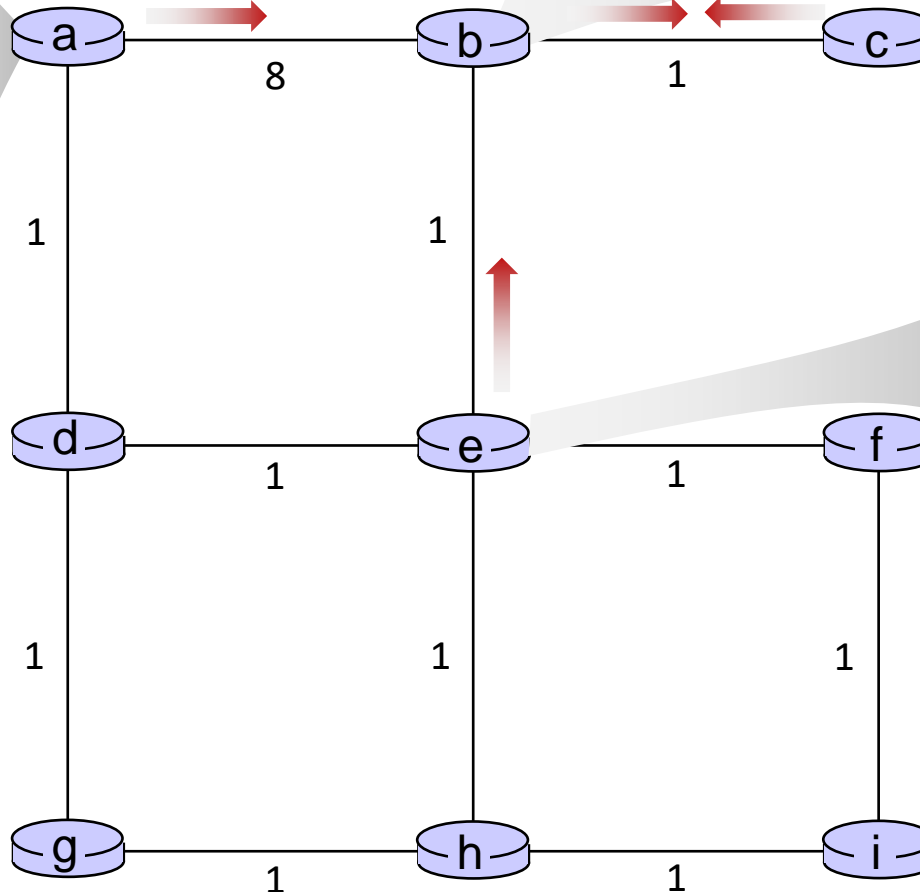
$D_b(a)=8$ $D_b(f)=\infty$
 $D_b(c)=1$ $D_b(g)=\infty$
 $D_b(d)=\infty$ $D_b(h)=\infty$
 $D_b(e)=1$ $D_b(i)=\infty$

DV in c:

$D_c(a)=\infty$
 $D_c(b)=1$
 $D_c(c)=0$
 $D_c(d)=\infty$
 $D_c(e)=\infty$
 $D_c(f)=\infty$
 $D_c(g)=\infty$
 $D_c(h)=\infty$
 $D_c(i)=\infty$

DV in e:

$D_e(a)=\infty$
 $D_e(b)=1$
 $D_e(c)=\infty$
 $D_e(d)=1$
 $D_e(e)=0$
 $D_e(f)=1$
 $D_e(g)=\infty$
 $D_e(h)=1$
 $D_e(i)=\infty$



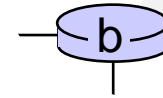
Distance vector example:



t=1

- c receives DVs from b computes:

$$\begin{aligned}D_c(a) &= \min\{c_{c,b} + D_b(a)\} = 1 + 8 = 9 \\D_c(b) &= \min\{c_{c,b} + D_b(b)\} = 1 + 0 = 1 \\D_c(d) &= \min\{c_{c,b} + D_b(d)\} = 1 + \infty = \infty \\D_c(e) &= \min\{c_{c,b} + D_b(e)\} = 1 + 1 = 2 \\D_c(f) &= \min\{c_{c,b} + D_b(f)\} = 1 + \infty = \infty \\D_c(g) &= \min\{c_{c,b} + D_b(g)\} = 1 + \infty = \infty \\D_c(h) &= \min\{c_{c,b} + D_b(h)\} = 1 + \infty = \infty \\D_c(i) &= \min\{c_{c,b} + D_b(i)\} = 1 + \infty = \infty\end{aligned}$$



1

compute

DV in b:

$D_b(a) = 8$	$D_b(f) = \infty$
$D_b(c) = 1$	$D_b(g) = \infty$
$D_b(d) = \infty$	$D_b(h) = \infty$
$D_b(e) = 1$	$D_b(i) = \infty$

DV in c:

$D_c(a) = \infty$
$D_c(b) = 1$
$D_c(c) = 0$
$D_c(d) = \infty$
$D_c(e) = \infty$
$D_c(f) = \infty$
$D_c(g) = \infty$
$D_c(h) = \infty$
$D_c(i) = \infty$

DV in c:

$D_c(a) = 9$
$D_c(b) = 1$
$D_c(c) = 0$
$D_c(d) = 2$
$D_c(e) = \infty$
$D_c(f) = \infty$
$D_c(g) = \infty$
$D_c(h) = \infty$
$D_c(i) = \infty$

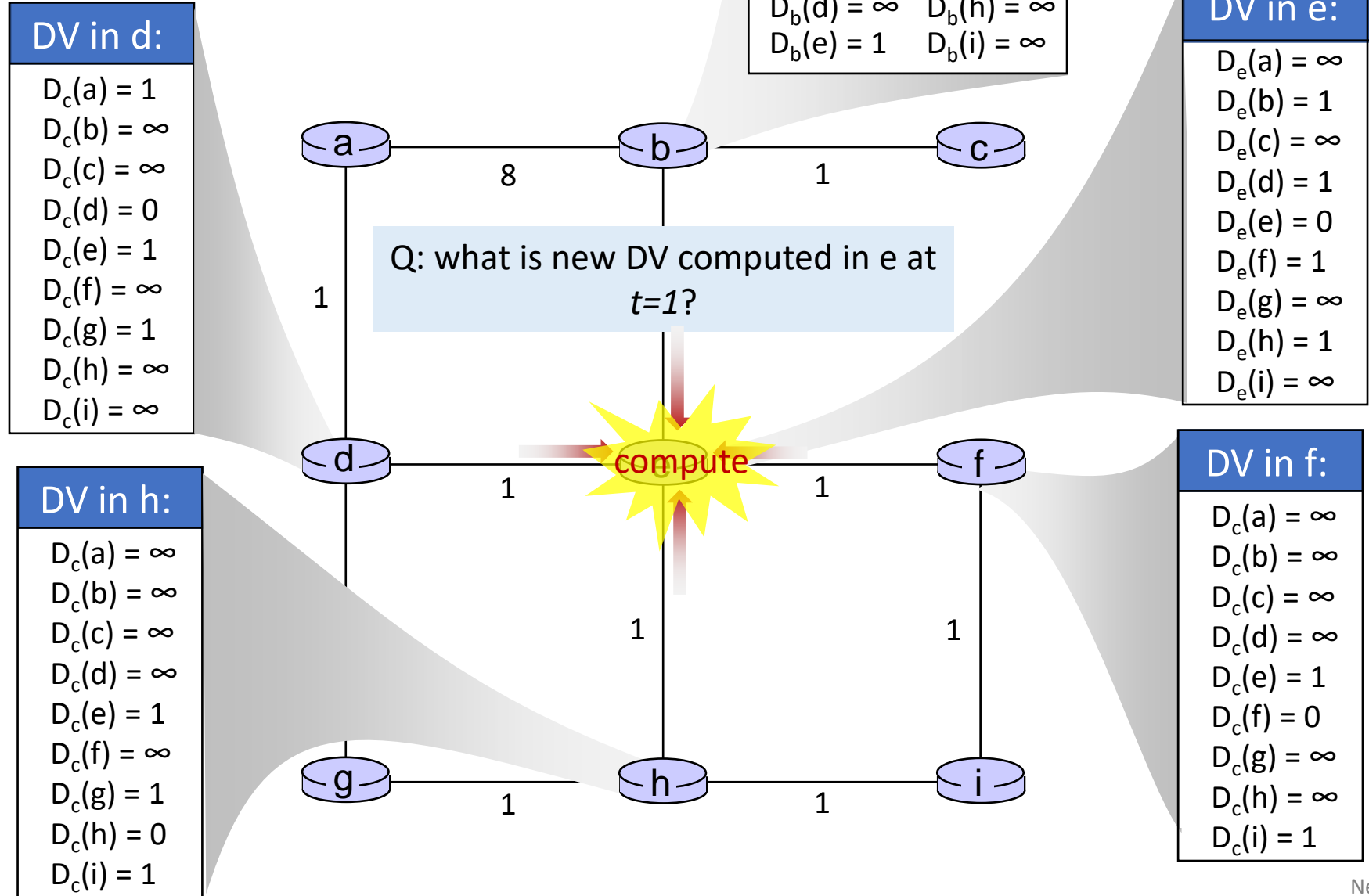
* Check out the online interactive exercises for more examples:
http://gaia.cs.umass.edu/kurose_ross/interactive/

Distance vector example:








t=1

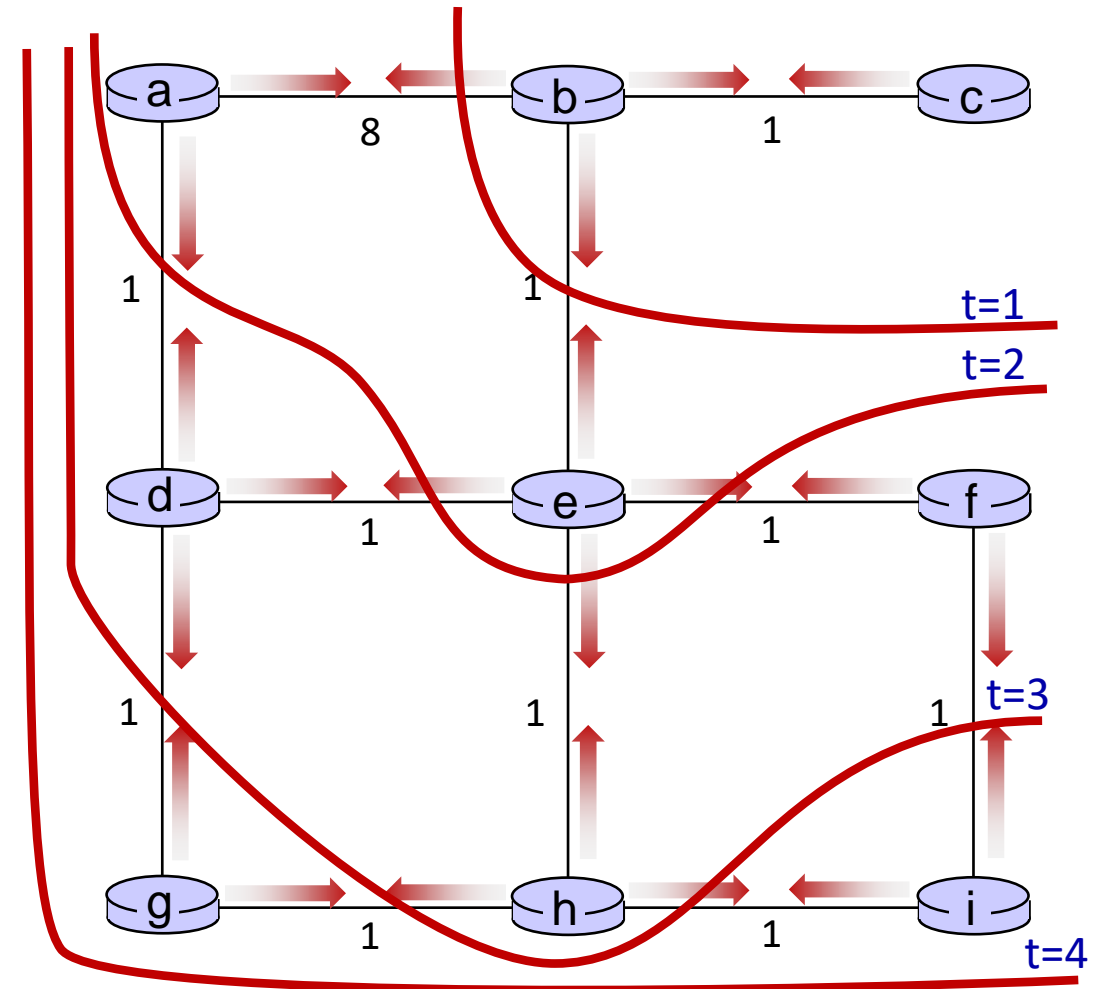
- e receives DVs from b, d, f, h



Distance vector: state information diffusion

Iterative communication, computation steps diffuses information through network:

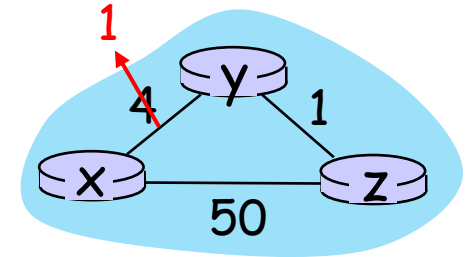
-  $t=0$ c's state at $t=0$ is at c only
-  $t=1$ c's state at $t=0$ has propagated to b, and may influence distance vector computations up to **1** hop away, i.e., at b
-  $t=2$ c's state at $t=0$ may now influence distance vector computations up to **2** hops away, i.e., at b and now at a, e as well
-  $t=3$ c's state at $t=0$ may influence distance vector computations up to **3** hops away, i.e., at b,a,e and now at c,f,h as well
-  $t=4$ c's state at $t=0$ may influence distance vector computations up to **4** hops away, i.e., at b,a,e, c, f, h and now at g,i as well



Distance vector: link cost changes

link cost changes:

- node detects local link cost change
- updates routing info, recalculates local DV
- if DV changes, notify neighbors



“good news
travels fast”

t_0 : y detects link-cost change, updates its DV, informs its neighbors.

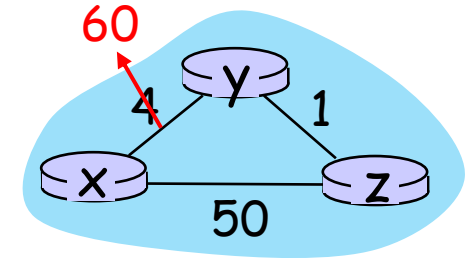
t_1 : z receives update from y, updates its table, computes new least cost to x, sends its neighbors its DV.

t_2 : y receives z's update, updates its distance table. y's least costs do *not* change, so y does *not* send a message to z.

Distance vector: link cost changes

link cost changes:

- node detects local link cost change
- “bad news travels slow” – count-to-infinity problem;



- y sees direct link to x has new cost 60, but z has said it has a path at cost of 5. So y computes “my new cost to x will be 6, via z); notifies z of new cost of 6 to x.
 - z learns that path to x via y has new cost 6, so z computes “my new cost to x will be 7 via y), notifies y of new cost of 7 to x.
 - y learns that path to x via z has new cost 7, so y computes “my new cost to x will be 8 via y), notifies z of new cost of 8 to x.
 - z learns that path to x via y has new cost 8, so z computes “my new cost to x will be 9 via y), notifies y of new cost of 9 to x.
 - ...
- see text for solutions. *Distributed algorithms are tricky!*

Comparison of LS and DV algorithms

message complexity

LS: n routers, $O(n^2)$ messages sent

DV: exchange between neighbors;
convergence time varies

speed of convergence

LS: $O(n^2)$ algorithm, $O(n^2)$ messages

- may have oscillations

DV: convergence time varies

- may have routing loops
- count-to-infinity problem

robustness: what happens if router malfunctions, or is compromised?

LS:

- router can advertise incorrect *link* cost
- each router computes only its *own* table

DV:

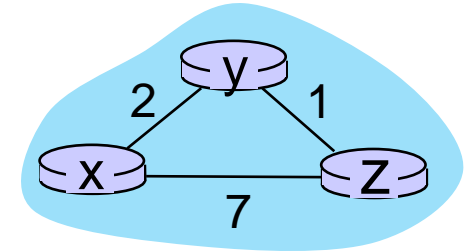
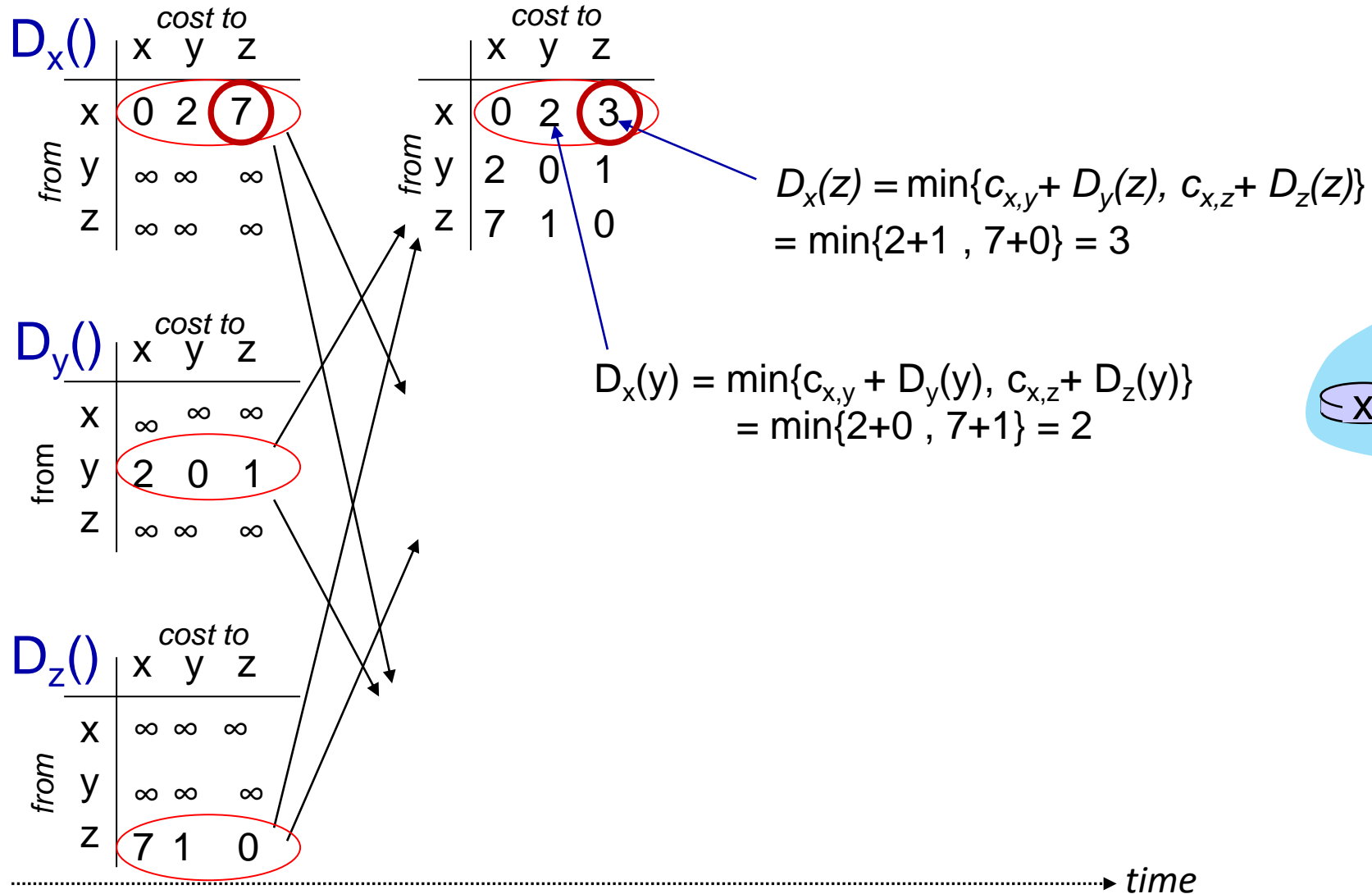
- DV router can advertise incorrect *path* cost (“I have a *really* low cost path to everywhere”): black-holing
- each router’s table used by others: error propagate thru network

کلا دو نوع پروتکل داریم:

پروتکل های مبتنی برای الگوریتم های link state یا LS و الگوریتم های Distance vector یا DV

توی این اسلاید مقایسه این دوتارو گفته از جنبه های مختلف

Distance vector: another example



Dx دیتابیس نود x مون است که شامل Distance vector نود x است ینی فاصله x تا همه مقصد ها که توی جدول نوشته و بعد اطلاعات همسایه هاشو باید داشته باشه که اینجا y, z است ینی Distance vector همسایه هاشو میخواد و اینجا چون در ابتدا است و اینا رو هنوز نداره بی نهایت گذاشته

و بعد دیتابیس y و Distance vector خودش و اطلاعات همسایه هاش و Distance vector همسایه هاش و بعد همین مورد برای دیتابیس z ... حالا iteration اولمون چیه؟

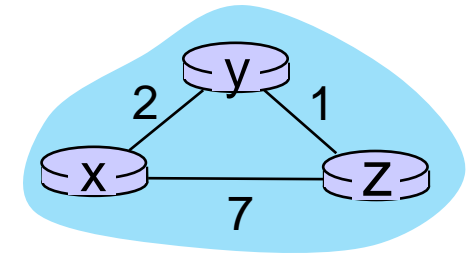
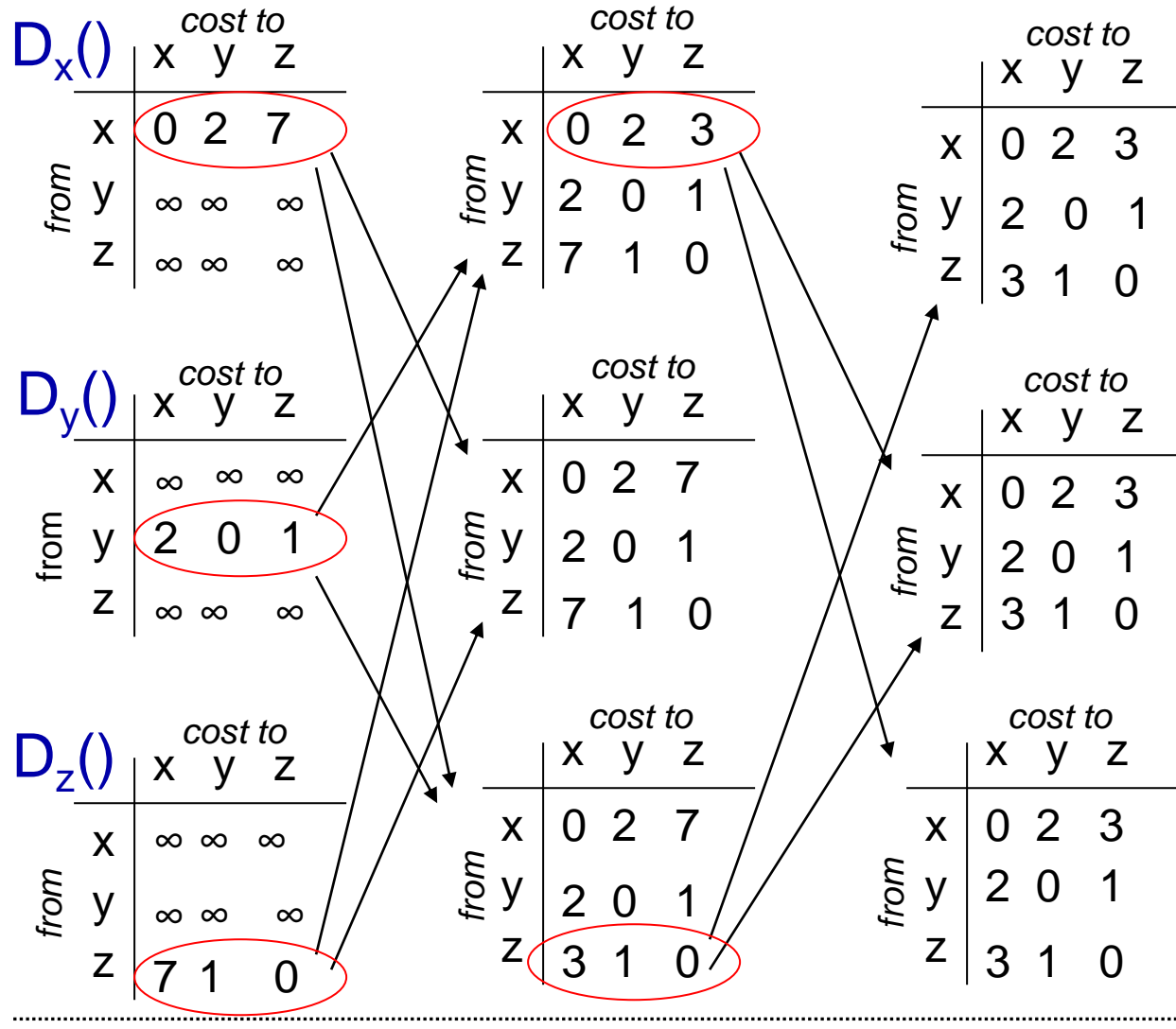
هر نود میاد Distance vector خودشو به همسایه هاش میده ینی بعد از یک iteration ، Distance vector نود y رو x داره و Distance vector نود z رو هم x داره که این همیشه اطلاعات محلی خودش و بعد براساس این ها و فاصله ای که خودش تا همسایه ها داره میاد Distance vector خودش رو تا مقصد ها اپدیت میکنه براساس اون فرمولی که گفتیم که توی اسلاید روبرو نوشته

و این اتفاق هایی که بالا گفتیم واسه y و z هم می افته که توی صفحه بعدی نوشته شده...

Distance vector: another example

توی iteration اول

توی iteration دوم



نکته: اگر تغییری دیگه روی Distance vector به وجود نیاد به این معنی است که دیگه پایدار شده است

نکته: در تمرین داشتیم که اگر برای یک نودی مثلا x هزینه یکی از مسیرهایش را عوض کنیم و در این حالت یک نود دیگه انتخاب شود پس اون نود x که مبدا ما بود باید این اپدیت رو به همسایه هایش اطلاع بدهد