Coordinated Attack

Distributed Systems

Ali Kamandi, PH.D.

School of Engineering Science

College of Engineering

University of Tehran

kamandi@ut.ac.ir

2024

مدل های خطا

- مدل خطا مشخص می‌کند که به چه طریقی اجزای سیستم ممکن است دچار خطا شوند. هر یک از الگوریتم‌ها با برخی از مدل‌های خطا سازگار هستند و از این جهت شناخت دقیق مدل‌های خطا ضرورت دارد.
- یک سیستم t-fault-tolerant نامیده می‌شود اگر با وجود حداکثر t جزء خطادار، بتواند هدف خود را محقق کند.
- MTBF یا همان Mean time between failures نشان دهنده زمان مورد انتظار تا بروز خطا می‌باشد.

Process failure models (Benign)

- **Fail-stop**

در این مدل، از یک لحظه به بعد پروسه‌ای که به درستی کار می‌کرده از کار می افتد. سایر پروسه‌ها می‌فهمند که این پروسه دچار اشکال شده است. این مدل یک مدل انتزاعی است و نحوه فهم دیگران از خطا می‌تواند کاملا متنوع باشد.

- **Crash:**

در این مدل یک پروسه که درست کار می‌کرده در یک لحظه از کار می‌افتد. نودهای دیگر از آن مطلع نمی‌شوند.

- **Receive omission:**

در این مدل یک پروسه درست، به نحوی دچار خطا می‌شود که فقط برخی از پیام‌هایی را که برای آن ارسال شده‌است را دریافت می‌کند.

- **Send omission:**

پروسه فقط بعضی از پیام‌هایی را که باید ارسال کند، واقعا ارسال می‌کند.

- **General omission:**

پروسه دچار یکی یا هر دو خطای فوق می‌شود.

Process failure models (Byzantine)

- **Byzantine or malicious failure, with authentication**

  پروسه ممکن است هر رفتار دلخواهی را از خود نشان دهد. اما چنانچه ادعا کند که پیامی را از نود خاصی دریافت کرده است، این ادعا با مکانیزم authentication یا امضا قابل ارزیابی خواهد بود.

- **Byzantine or malicious failure**

  پروسه ممکن است هر رفتار دلخواهی نشان دهد و بر خلاف مدل ادعای دریافت پیام از یک نود سالم، قبل قابل ارزیابی نخواهد بود.

Communication failure models

- **Crash failure**

یک لینک سالم از یک لحظه، هیچ پیامی را به مقصد نمی رساند.

- **Omission failure**

لینک بعضی از پیام ها را به مقصد می رساند و بعضی را نمی رساند.

- **Byzantine failure**

لینک ممکن است هر رفتار دلخواهی را بروز دهد. مثلا پیامی ایجاد کند یا محتوای یک پیام را تغییر دهد.

علاوه بر خطاهای فوق خطای timing هم در مورد پروسه‌ها و هم لینک ارتباطی می‌تواند وجود داشته باشد که مختص سیستم‌های synchronized است. به این معنی که یک لینک پیام‌ها را کندتر یا سریع‌تر از آنچه باید، انتقال دهد.

The coordinated attack problem

در مساله حمله هماهنگ چند ژنرال قرار است با نیروهایشان به یک هدف حمله کنند. در صورتی که همه به صورت هماهنگ حمله کنند، نتیجه موفقیت آمیز خواهد بود و در غیر اینصورت همگی نابود خواهند شد.

هر یک از ژنرال‌ها ذهنیت خود را از آمادگی نیروهایش دارد.

ژنرال‌ها در مکان‌های مختلفی مستقر شده‌اند. ارتباط بین ژنرال‌ها از طریق پیک‌هایی است که ممکن است اسیر یا کشته شوند. با توجه به اینکه راه ارتباطی بین آنها غیر قابل اطمینان است، ژنرال‌ها باید به طریقی هماهنگ با هم عمل کنند. یا همگی حمله کنند یا هیچ کدام. اولویت با حمله می‌باشد.

The coordinated attack problem

در صورتی که همه پیک ها قابل اطمینان بودند، کافی بود هر یک از ژنرال‌ها پیامی را به سایر ژنرال‌ها می‌فرستاد و از تصمیم آن ها سوال می‌کرد. سپس بعد از چند مرحله (قطر گراف ارتباطی)، همه ژنرال‌ها از همه تصمیمات با خبر می‌شدند. به عبارتی اگر همه آماده بودند، حمله آغاز می‌شد.

این مساله همان مساله distributed commit در دیتابیس‌های توزیع شده است.

در صورتی که امکان گم شدن پیک یا پیام وجود داشته باشد، این روش قابل اجرا نیست.

اثبات می‌شود که هیچ الگوریتمی وجود ندارد که همیشه بتواند این مساله را حل کند.

مدل مسئله

- n نود که از 1 تا n شماره‌گذاری شده‌اند.
- گراف ارتباطی بدون جهت می‌باشد.
- هر نود از کل گراف مطلع است.
- تصمیم هر نود 0 یا 1 است. 1 به معنی حمله یا کامیت خواهد بود.

شرایط تصمیم‌گیری

- **Safety:**
  - **Agreement**: no two processes decide on different values.
  - **Validity**:

اگر همه نودها با 0 شروع کردند، نتیجه نهایی باید 0 باشد.

اگر همه با 1 شروع کردند و هیچ پیامی گم نشد، نتیجه نهایی باید 1 باشد.

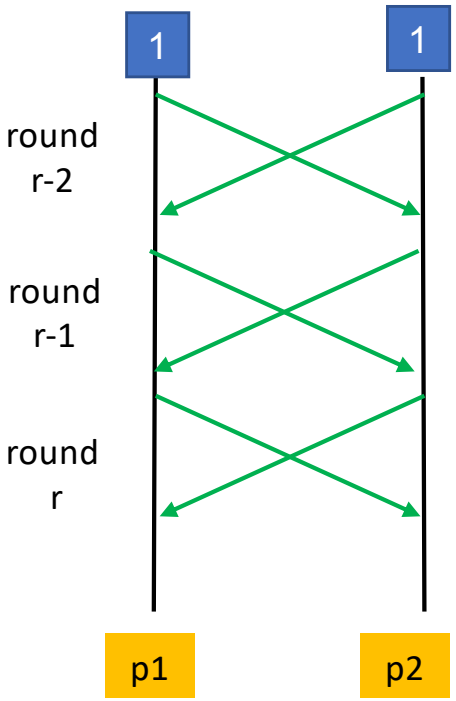
- **Liveness:**
  - **Termination**: All processes eventually decide.

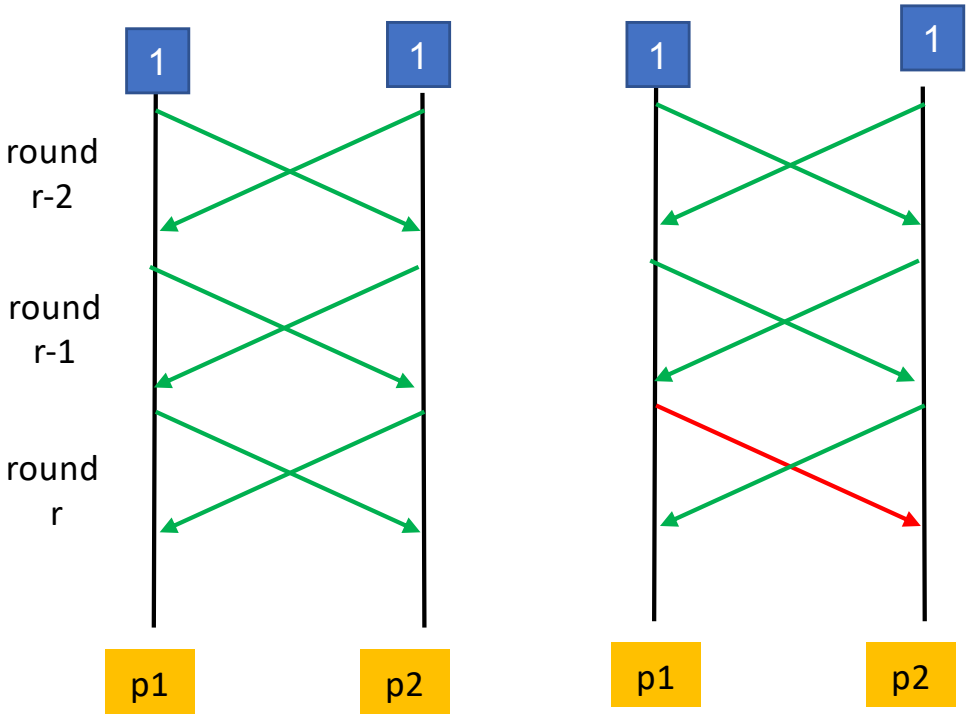
- **Fault Tolerance**

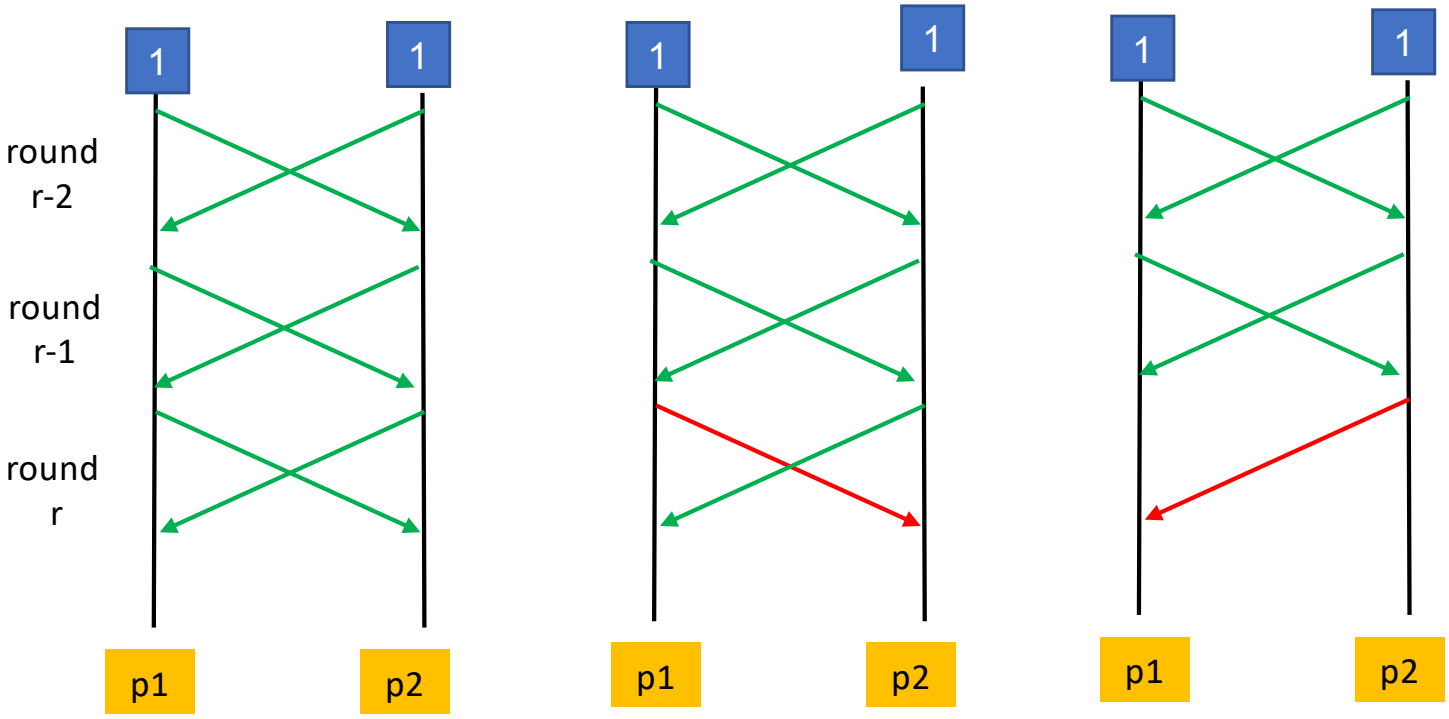
No deterministic consensus protocol provides all three of safety, liveness, and fault tolerance in an asynchronous system.

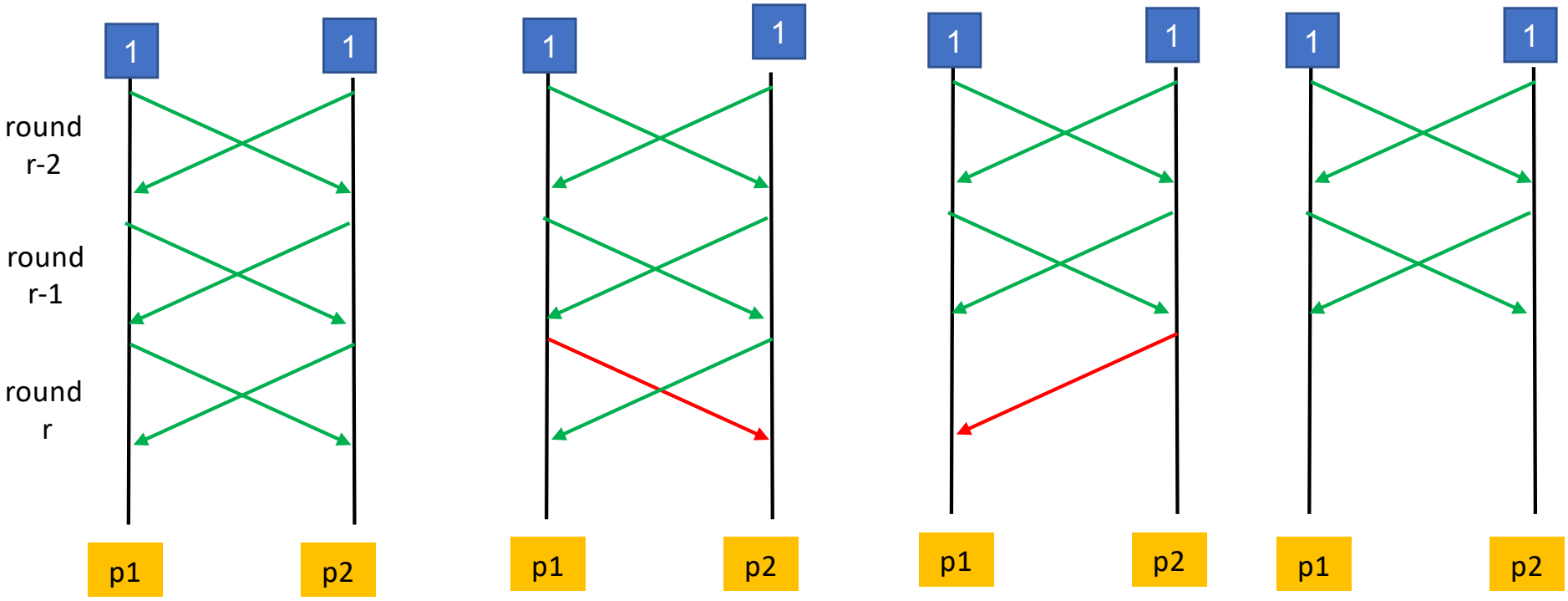
Theorem: Let G be the graph consisting of nodes 1 and 2 connected by a single edge. Then there is no algorithm that solves the coordinated attack problem on G.

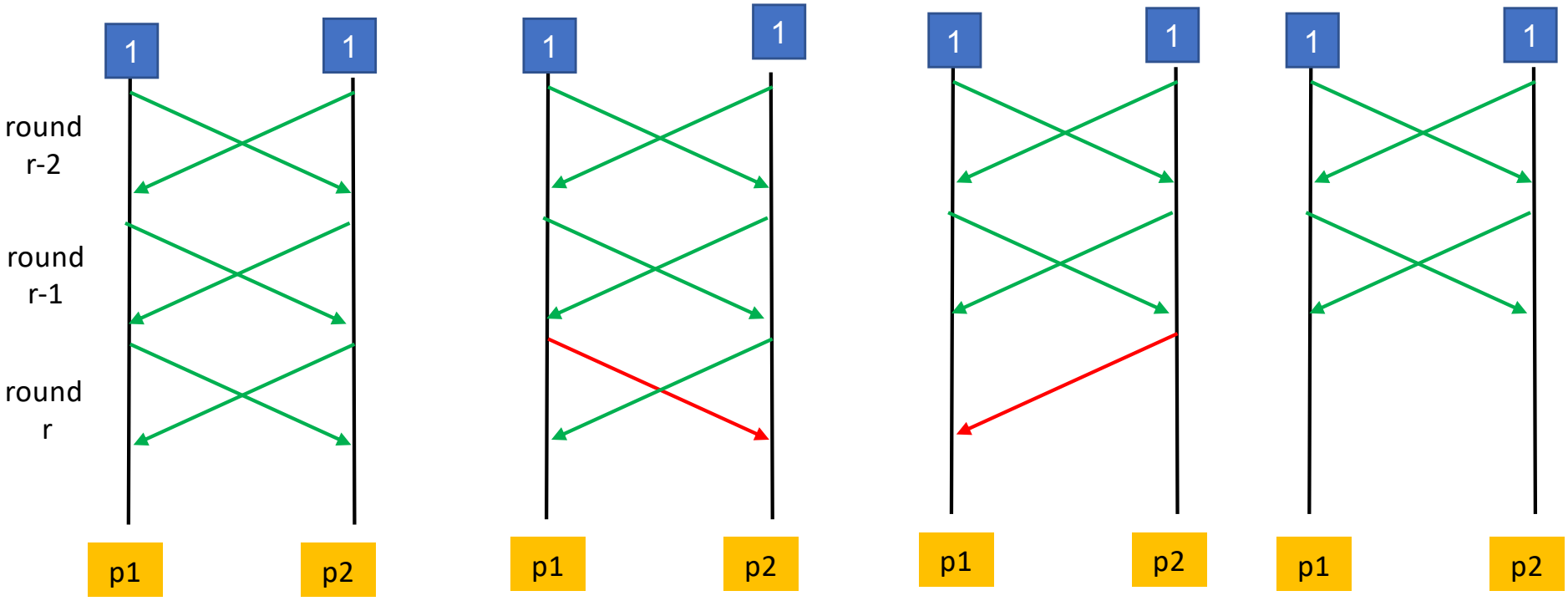
Proof: by **contradiction**.


1
1
round
r-2
round
r-1
round
r
p1
p2


1
1
round
r-2
round
r-1
round
r
p1
p2
1
1
p1
p2


round
r-2
round
r-1
round
r
1
1
p1
p2
1
1
p1
p2
1
1
p1
p2


round
r-2
round
r-1
round
r
1
1
p1
p2
1
1
p1
p2
1
1
p1
p2
1
1
p1
p2

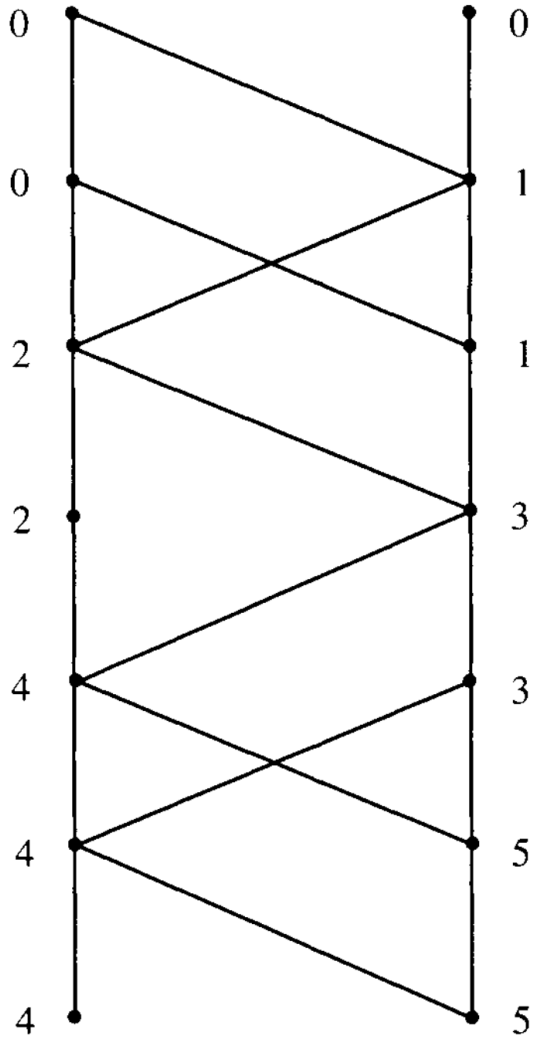


Starting from $\alpha 1$, we now construct a series of executions, each of them **indistinguishable** from its predecessor in the series with respect to one of the processes; it will follow that all of these executions must have the same decision value.

Randomized Algorithm

A simplifying assumption is that network is **complete**, although a **strongly-connected** network with *r* greater than or equal to the diameter also works.

Information Level



Randomized Algorithm

- First part: tracking information levels
  - Each process tracks its "information level," initially 0. The state of a process consists of a vector of (input, information-level) pairs for all processes in the system. Initially this is (my-input, 0) for itself and $(\perp, -1)$ for everybody else.
  - Every process sends its entire state to every other process in every round.
  - Upon receiving a message m, process i stores any inputs carried in m and, for each process j, sets $\mathsf{level}_i[j]$ to $\max(\mathsf{level}_i[j], \mathsf{level}_m[j])$. It then sets its own information level to $\min_j(\mathsf{level}_i[j]) + 1$.
- Second part: deciding the output
  - Process 1 chooses a random key value uniformly in the range $[1, r]$.
  - This key is distributed along with $\mathsf{level}_i[1]$, so that every process with $\mathsf{level}_i[1] \geq 0$ knows the key.
  - A process decides 1 at round r if and only if it knows the key, its information level is greater than or equal to the key, and all inputs are 1.

Randomized Algorithm

$states_i$:

$rounds \in \mathbb{N}$, initially 0
$decision \in \{unknown, 0, 1\}$, initially $unknown$
$key \in [1, r] \cup undefined$, initially $undefined$
for every j, $1 \leq j \leq n$:
    $val(j) \in \{0, 1, undefined\}$; initially $val(i)$ is i's initial value and
        $val(j) = undefined$ for all $j \neq i$
    $level(j) \in [-1, r]$; initially $level(i) = 0$ and $level(j) = -1$ for all $j \neq i$

Randomized Algorithm

$rand_i$:

if $i = 1$ and $rounds = 0$ then $key := random$

$msgs_i$:

send (L, V, key) to all j, where L is the *level* vector and V is the *val* vector

$trans_i$:

$rounds := rounds + 1$
let (L_j, V_j, k_j) be the message from j, for each j from which a message arrives
if, for some j, $k_j \neq undefined$ then $key := k_j$
for all $j \neq i$ do
    if, for some i', $V_{i'}(j) \neq undefined$ then $val(j) := V_{i'}(j)$
    if, for some i', $L_{i'}(j) > level(j)$ then $level(j) := \max_{i'} \{L_{i'}(j)\}$
$level(i) := 1 + \min \{level(j) : j \neq i\}$
if $rounds = r$ then
    if $key \neq undefined$ and $level(i) \geq key$ and $val(j) = 1$ for all j then
        $decision := 1$
    else $decision := 0$

Reference

- **Nancy Lynch, Chapter 5**