Protokoly jako komunikující automaty

Popisy potvrzovacích protokolů

Ondřej Votava

23. 3. 2011

Obsah

Následuje

Definice

$$FSM = (\mathbf{\Sigma}, \mathbf{S}, \delta, s, \mathbf{F})$$

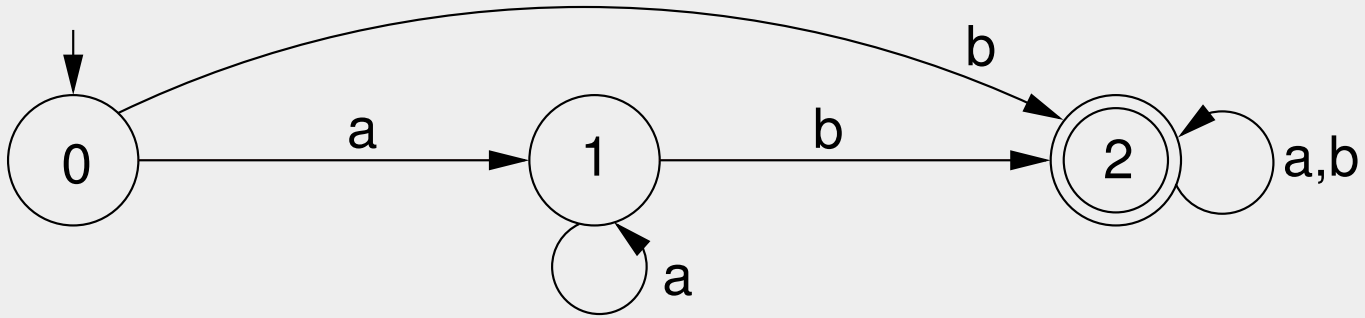
- $\mathbf{\Sigma}$ – konečná množina vstupních symbolů — abeceda
- $\mathbf{S}$ – konečná množina stavů
- δ – přechodová funkce $\delta : \mathbf{S} \times \mathbf{\Sigma} \to \mathbf{S}$
  - definuje chování automatu
  - podle vstupního symbolu určí následující stav
- s – počáteční stav, $s \in \mathbf{S}$
- $\mathbf{F}$ – množina koncových (přijímacích) stavů, $\mathbf{F} \subset \mathbf{S}$

Tabulka + text

Stav	Vstup	Další stav
0	a	1
0	b	2
1	a	1
1	b	2
2	a,b	2

- $\mathbf{\Sigma} = \{a, b\}$
- $\mathbf{S} = \{0, 1, 2\}$
- $s = 0$
- $\mathbf{F} = \{2\}$

Obrázek



Definice

$$FST = (\mathbf{\Sigma}, \mathbf{\Gamma}, \mathbf{S}, \delta, s, \omega)$$

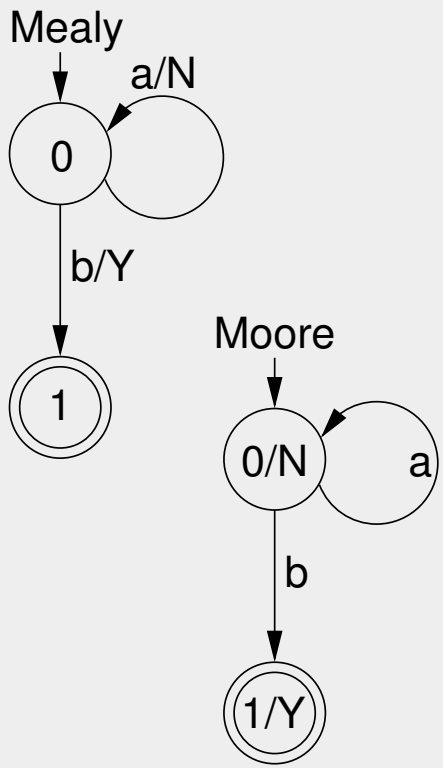
- $\mathbf{\Sigma}$ – konečná množina vstupních symbolů — abeceda
- $\mathbf{\Gamma}$ – konečná množina výstupních symbolů
- $\mathbf{S}$ – konečná množina stavů
- δ – přechodová funkce $\sigma : \mathbf{S} \times \mathbf{\Sigma} \to \mathbf{S}$
- s – počáteční stav, $s \in \mathbf{S}$
- ω – výstupní funkce
  - **Mealyho automat** – $\omega : \mathbf{S} \times \mathbf{\Sigma} \to \mathbf{\Gamma}$
  - **Mooreův automat** – $\omega : \mathbf{S} \to \mathbf{\Gamma}$

Tabulka

Stav	Vstup	Výstup	Další
0	a	N	0
0	b	Y	1

- $\mathbf{\Sigma} = \{a, b\}$
- $\mathbf{\Gamma} = \{N, Y\}$
- $\mathbf{S} = \{0, 1\}$
- $s = 0$

Obrázek



Následuje

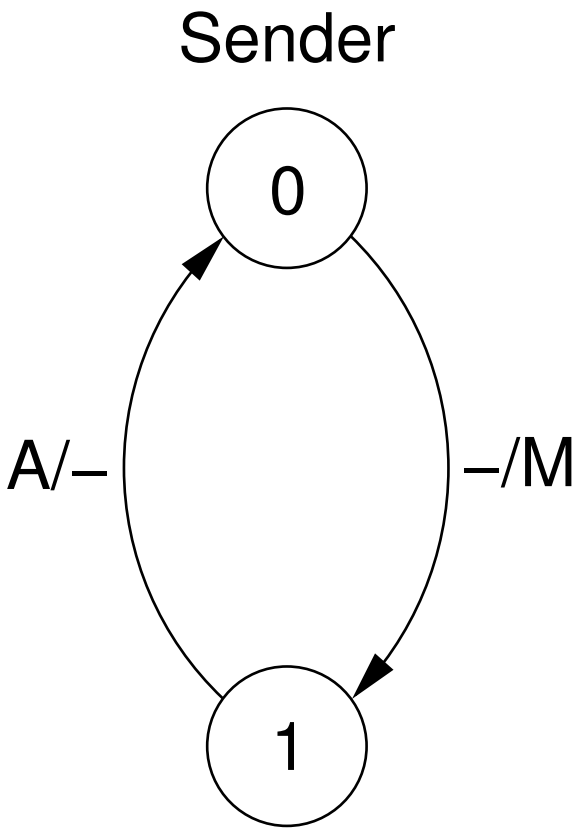
cs.wikipedia.org

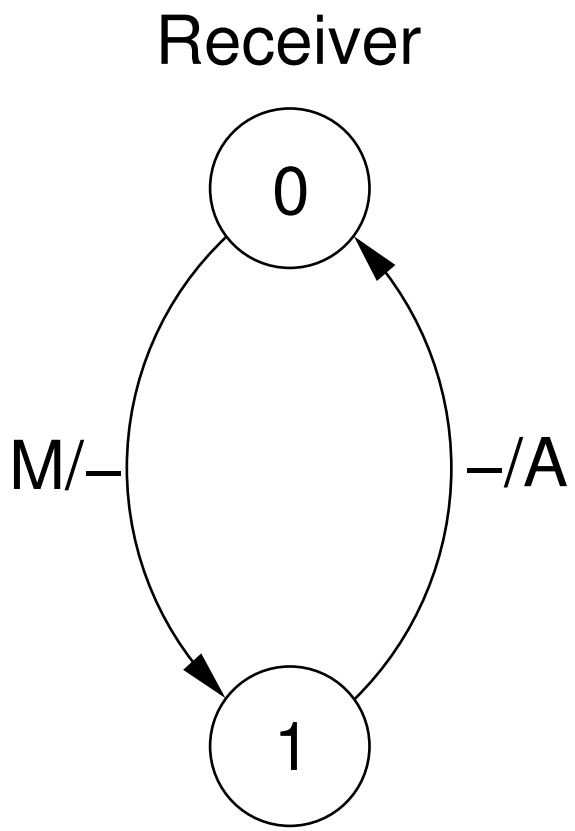
- konvence nebo standard, podle kterého probíhá elektronická komunikace a přenos dat mezi dvěma koncovými body
- protokol definuje pravidla řídící syntaxi, sémantiku a synchronizaci vzájemné komunikace

Gerard Holzmann – Design and validation of computer protocols

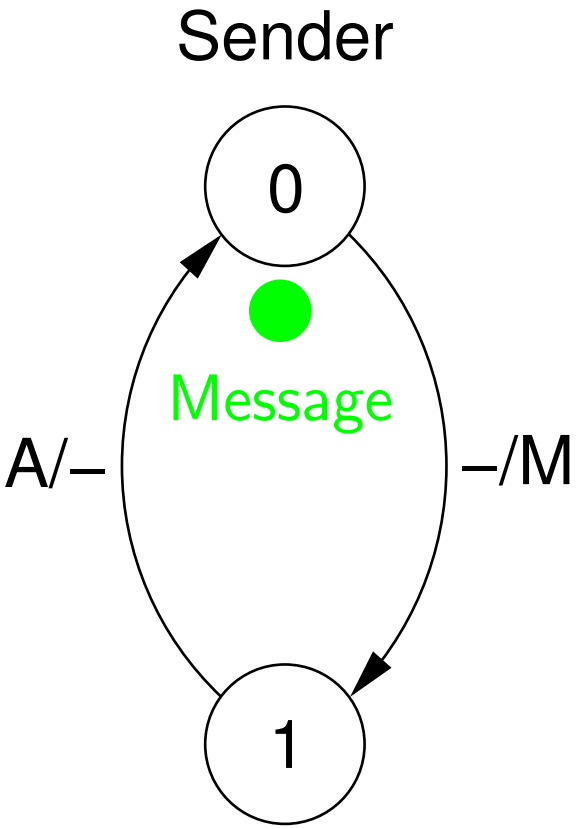
Protokoly jsou množinou pravidel, jež určují jak mají komunikovat souběžně běžící procesy v distribuovaných systémech.

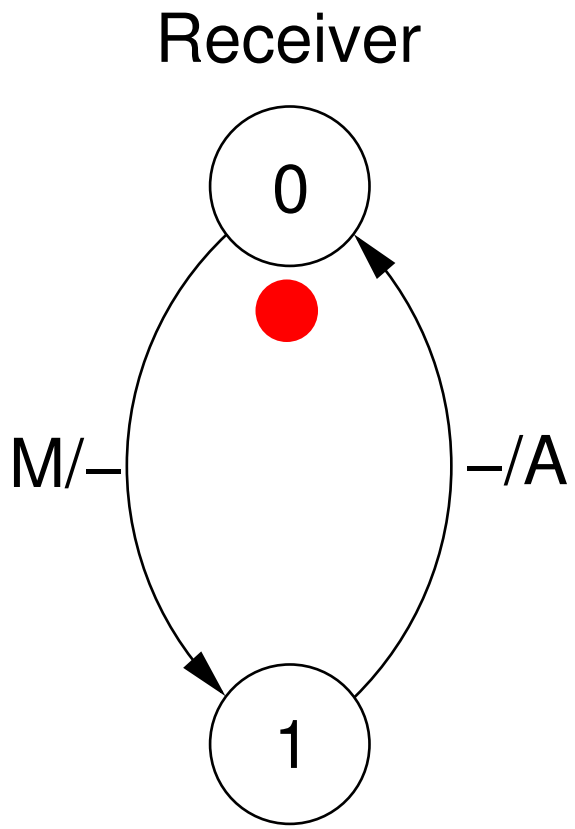
Jednoduchý potvrzovací protokol



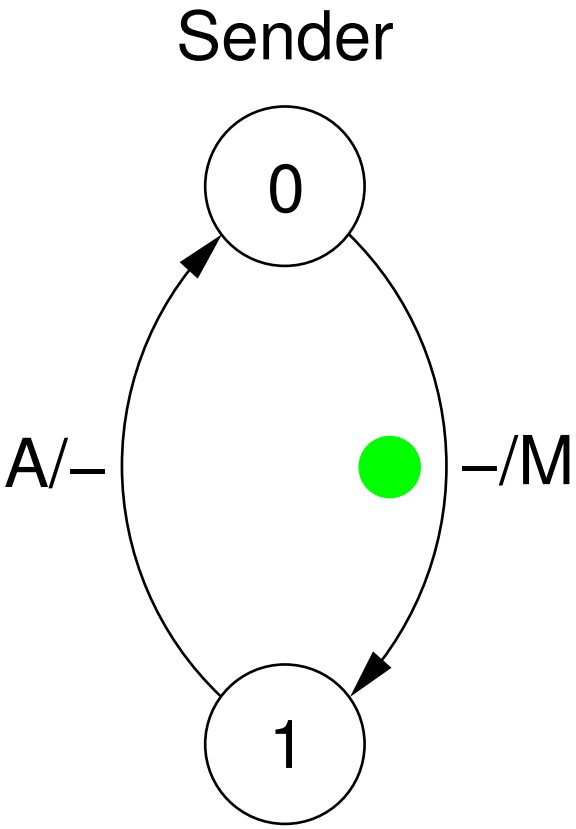


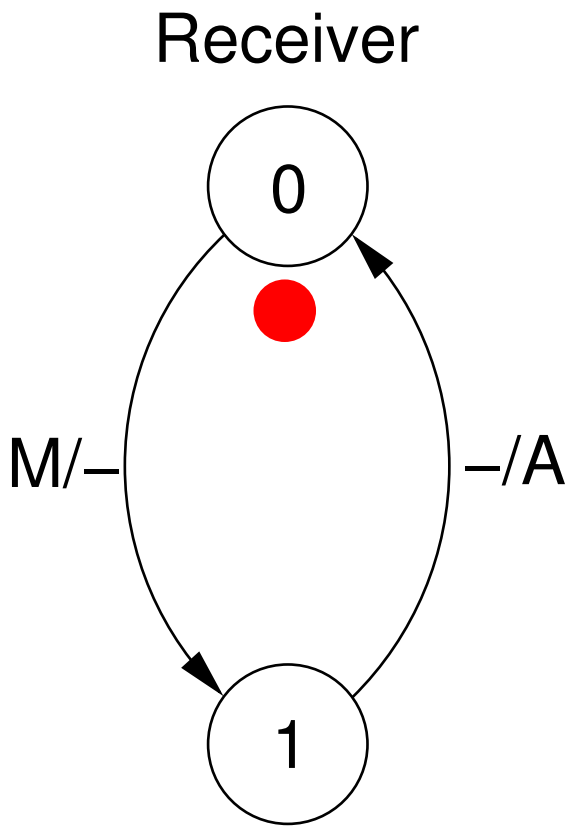
Jednoduchý potvrzovací protokol



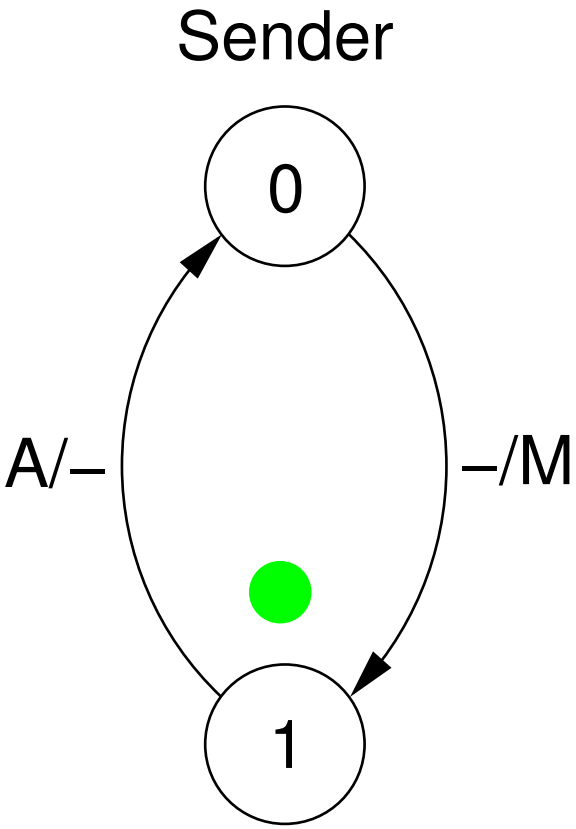


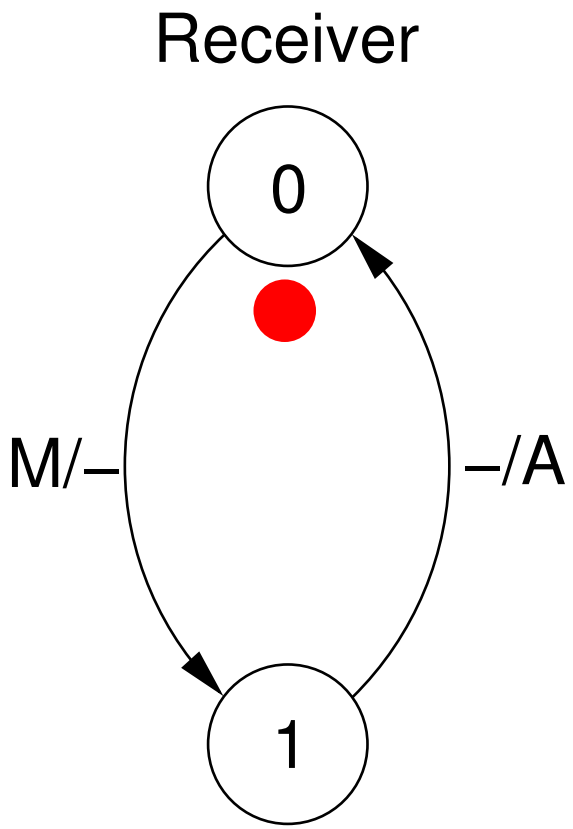
Jednoduchý potvrzovací protokol



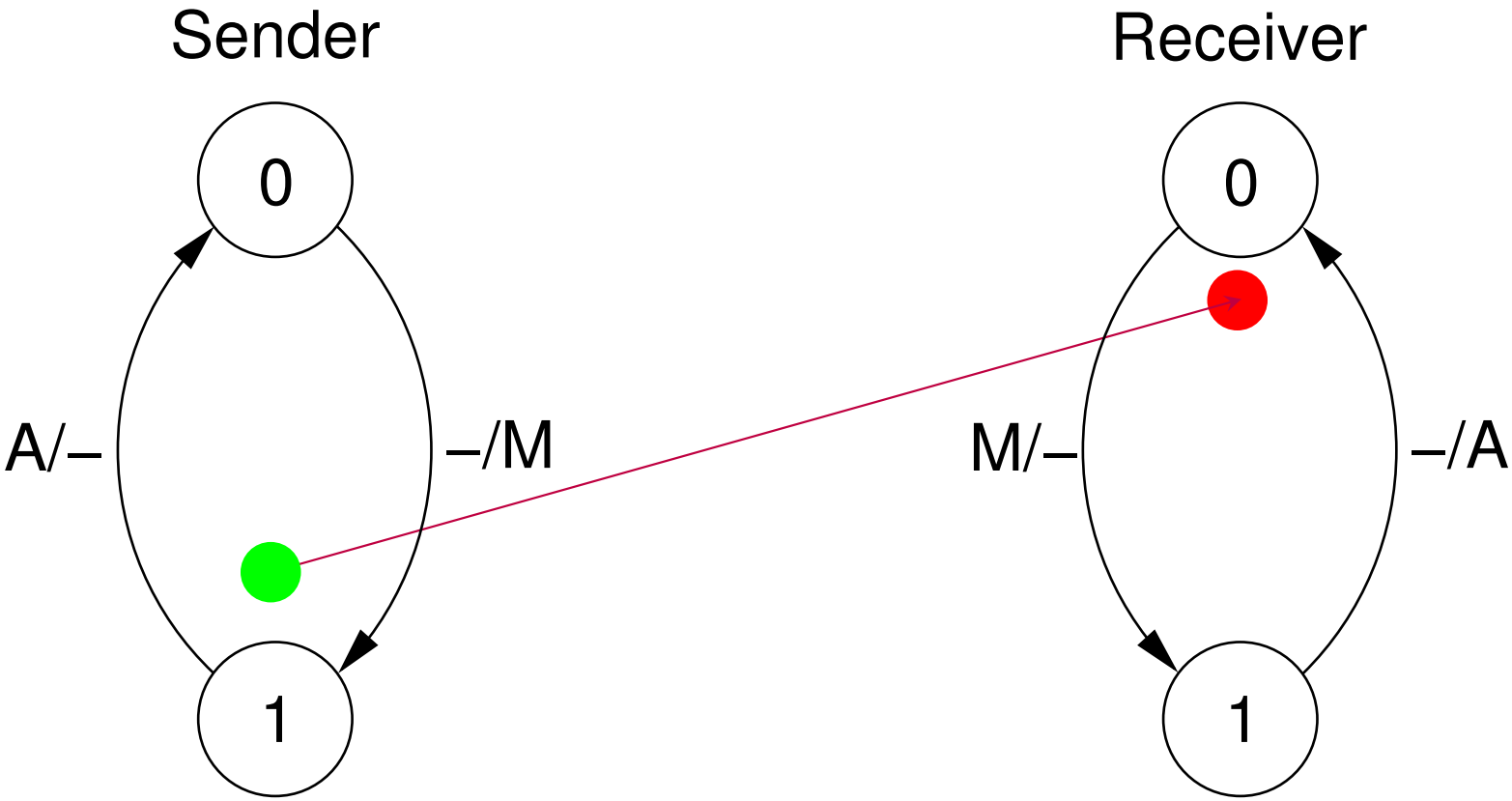


Jednoduchý potvrzovací protokol

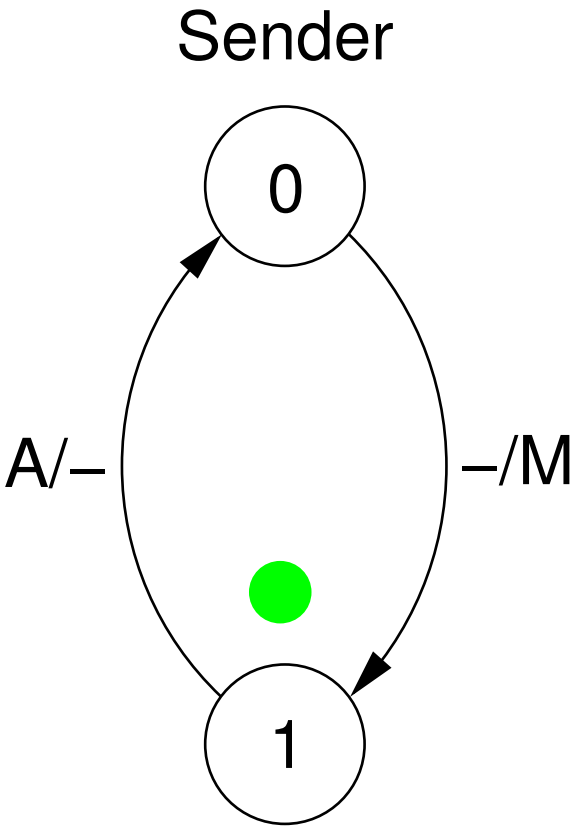


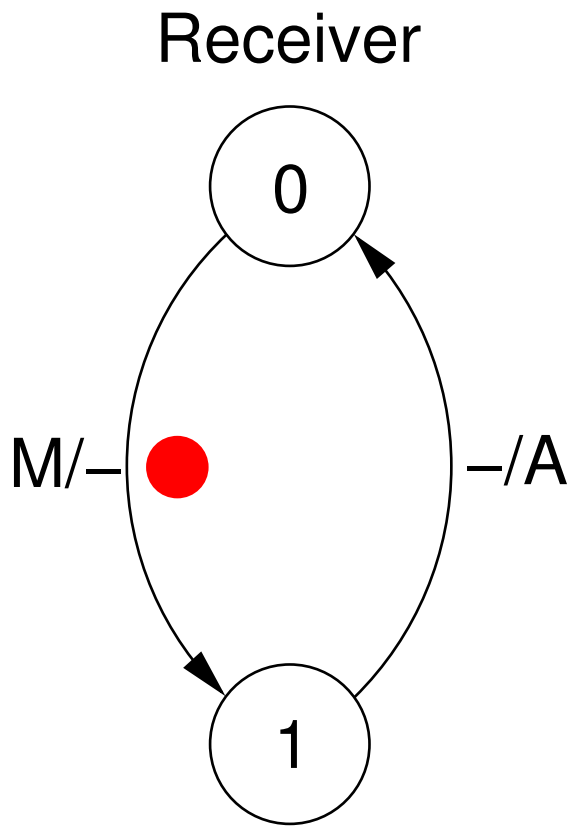


Jednoduchý potvrzovací protokol

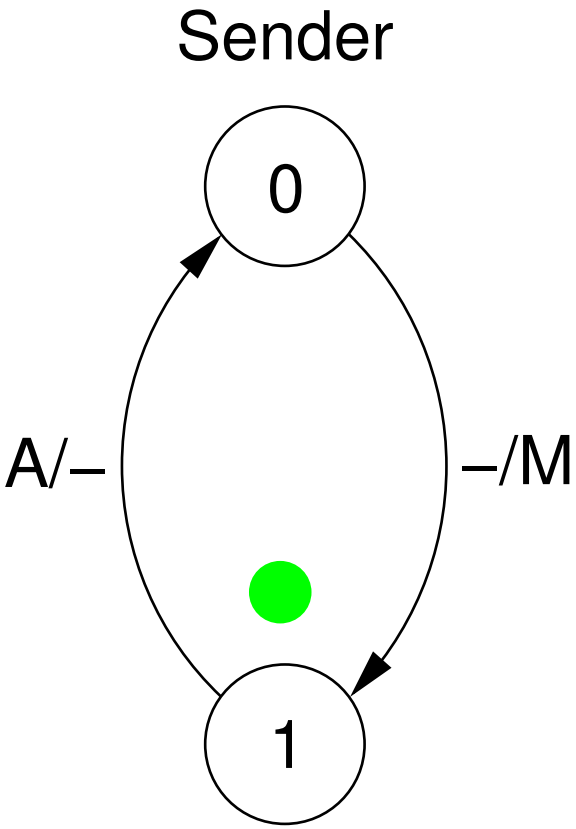


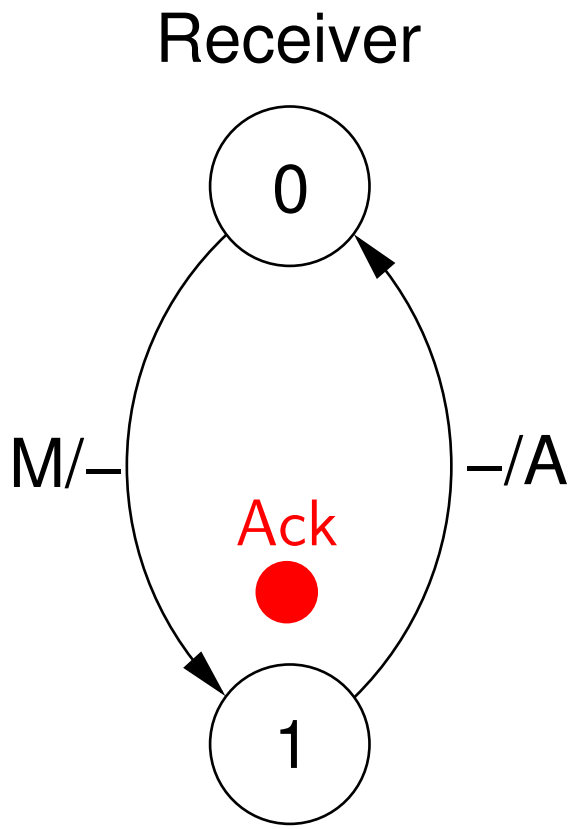
Jednoduchý potvrzovací protokol



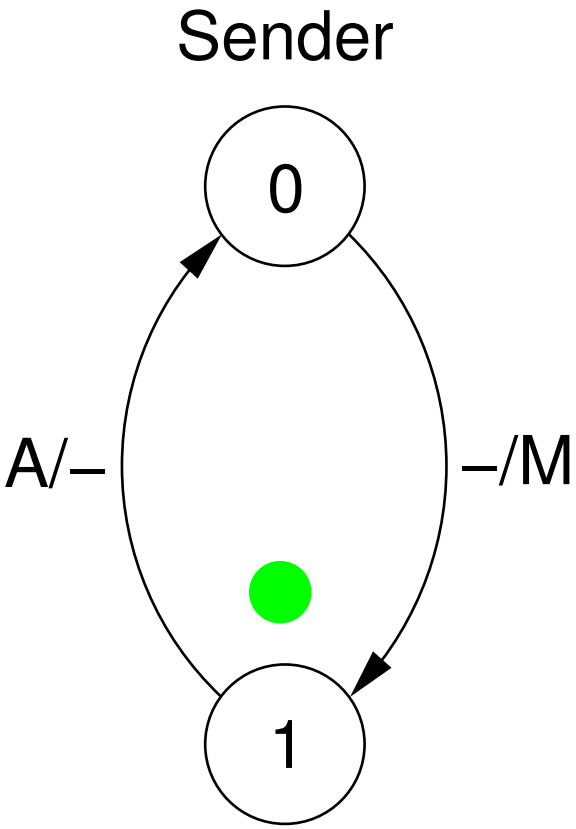


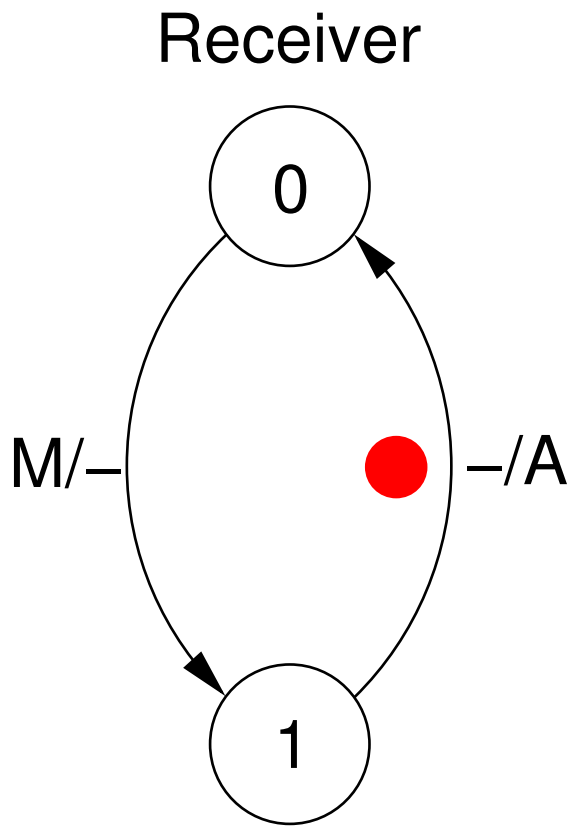
Jednoduchý potvrzovací protokol



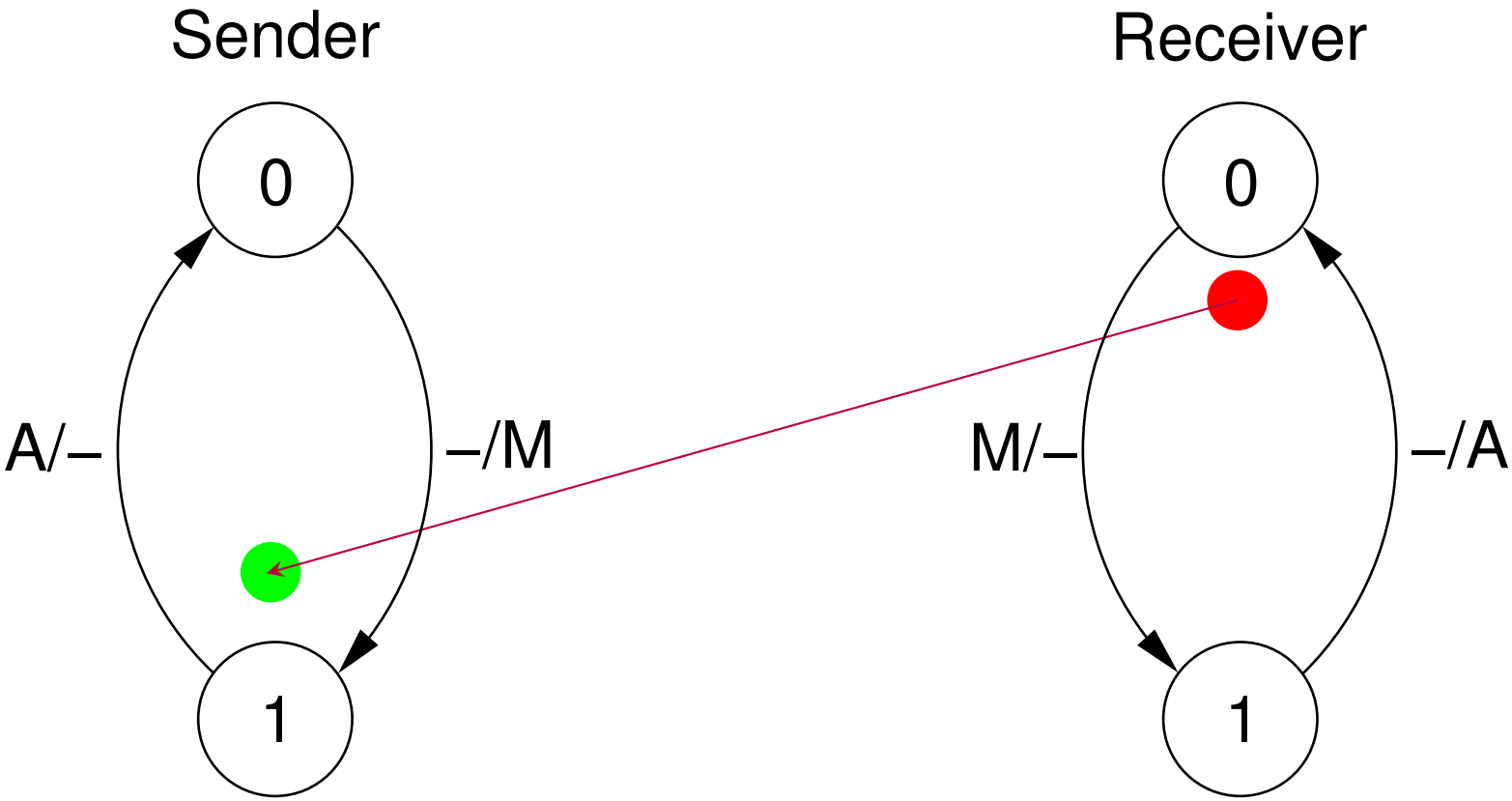


Jednoduchý potvrzovací protokol

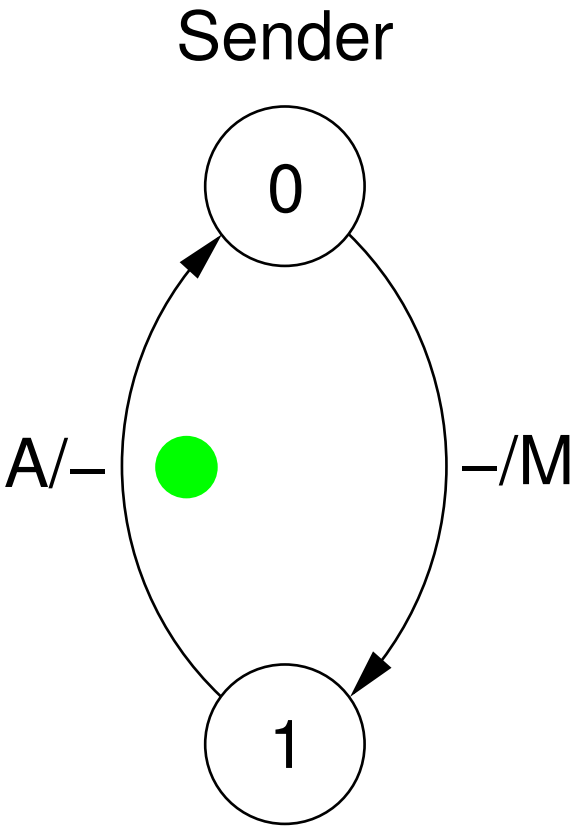


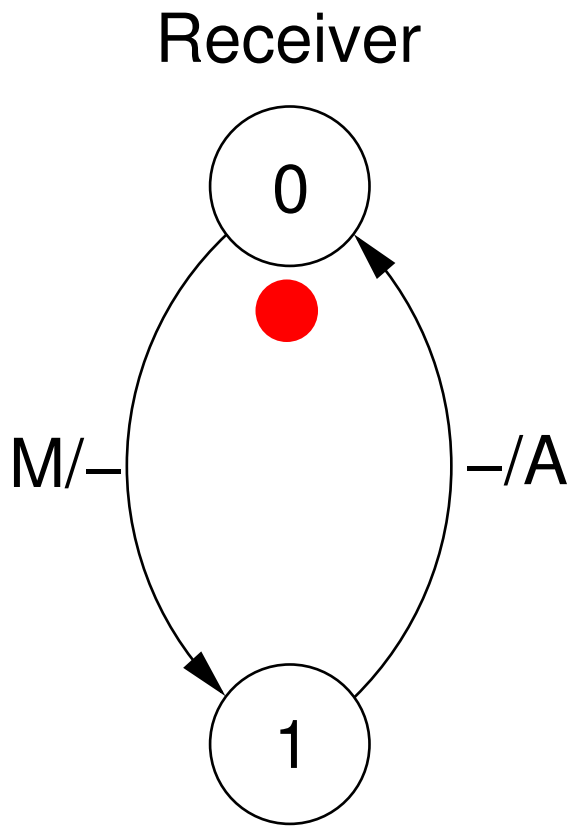


Jednoduchý potvrzovací protokol

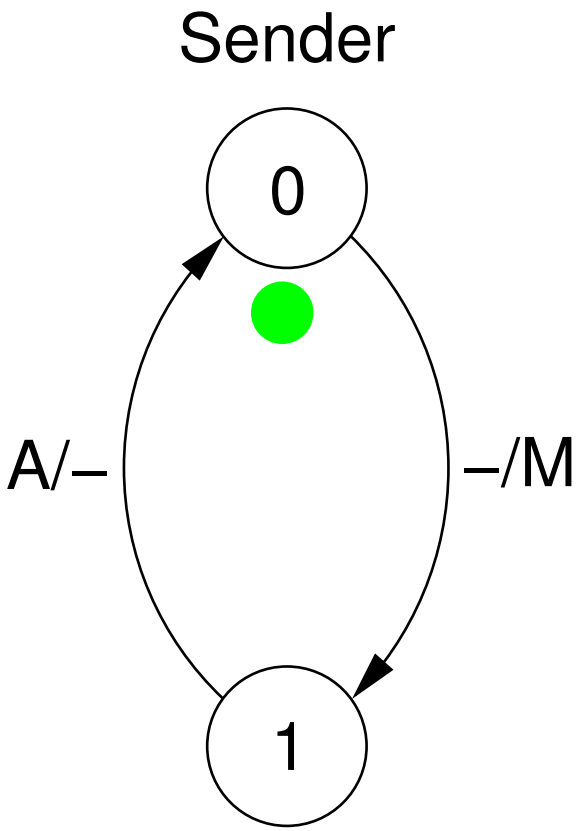


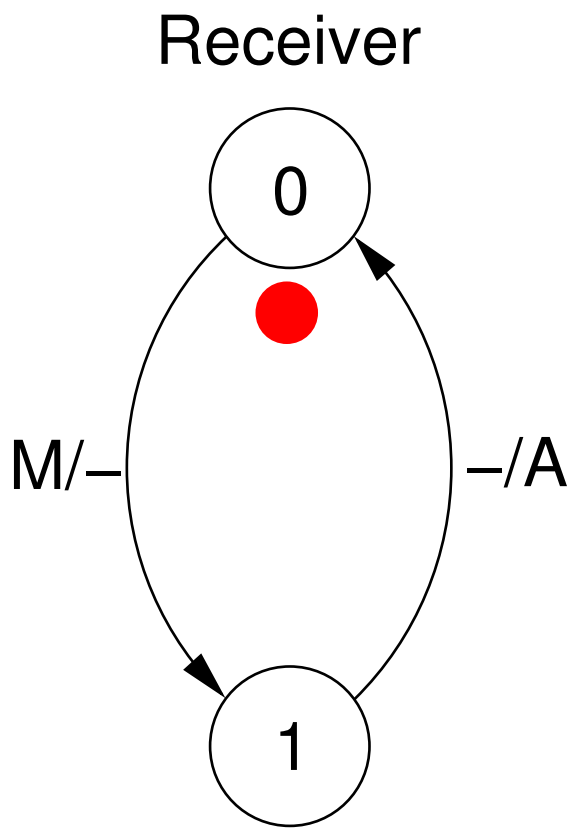
Jednoduchý potvrzovací protokol





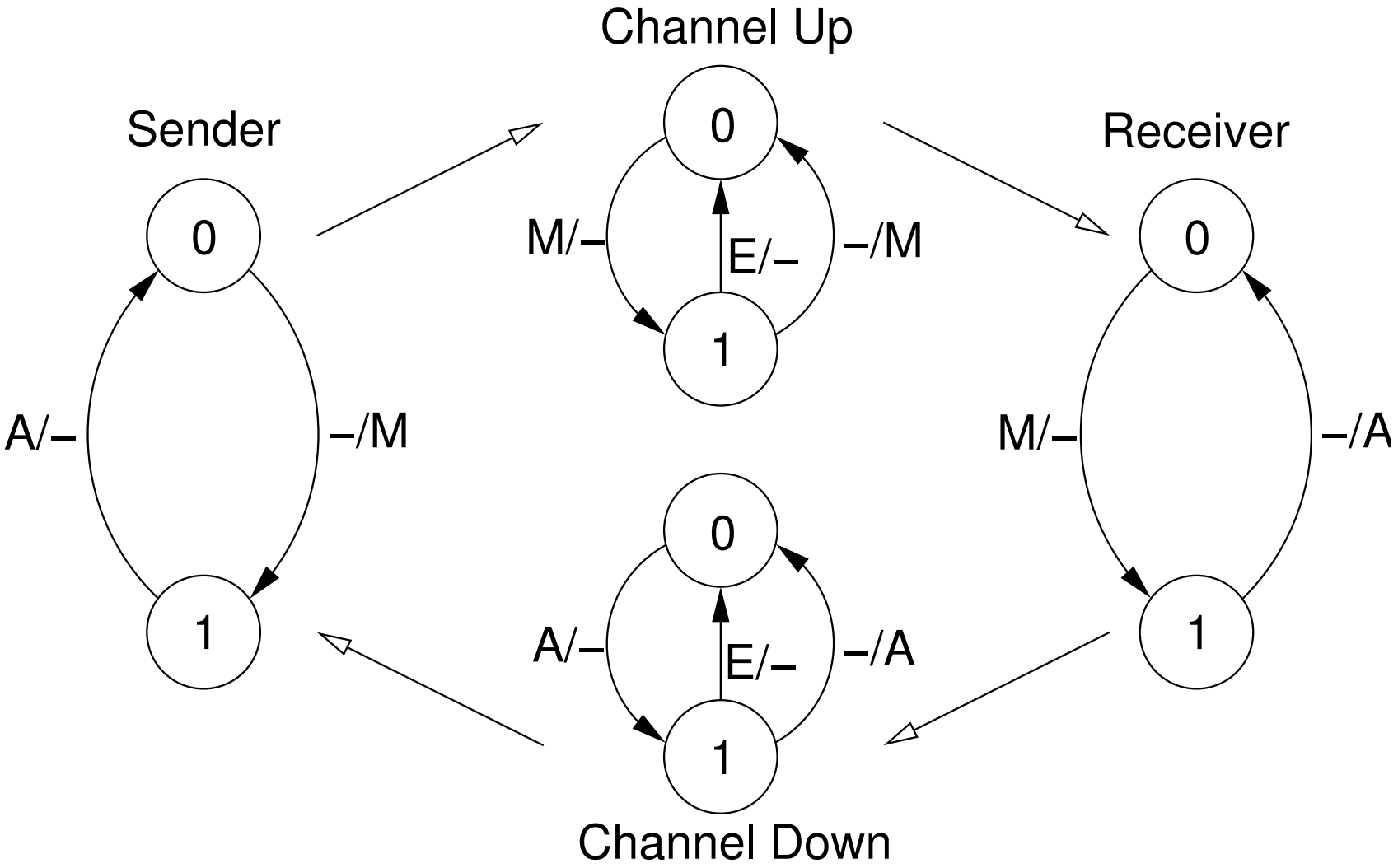
Jednoduchý potvrzovací protokol





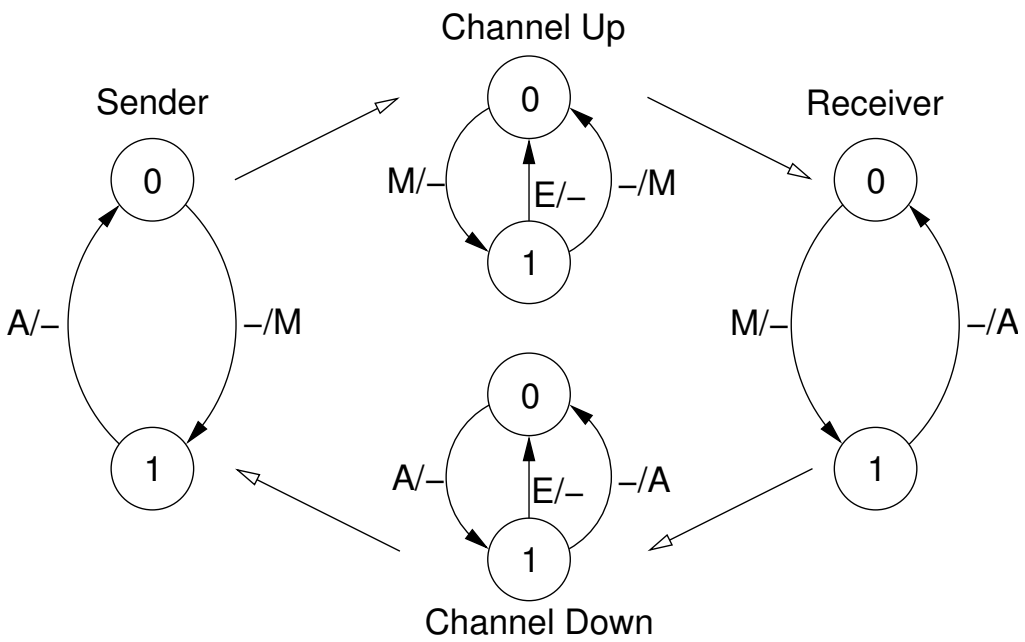
Jednoduchý potvrzovací protokol

S kanálem tvořícím chyby



Jednoduchý potvrzovací protokol

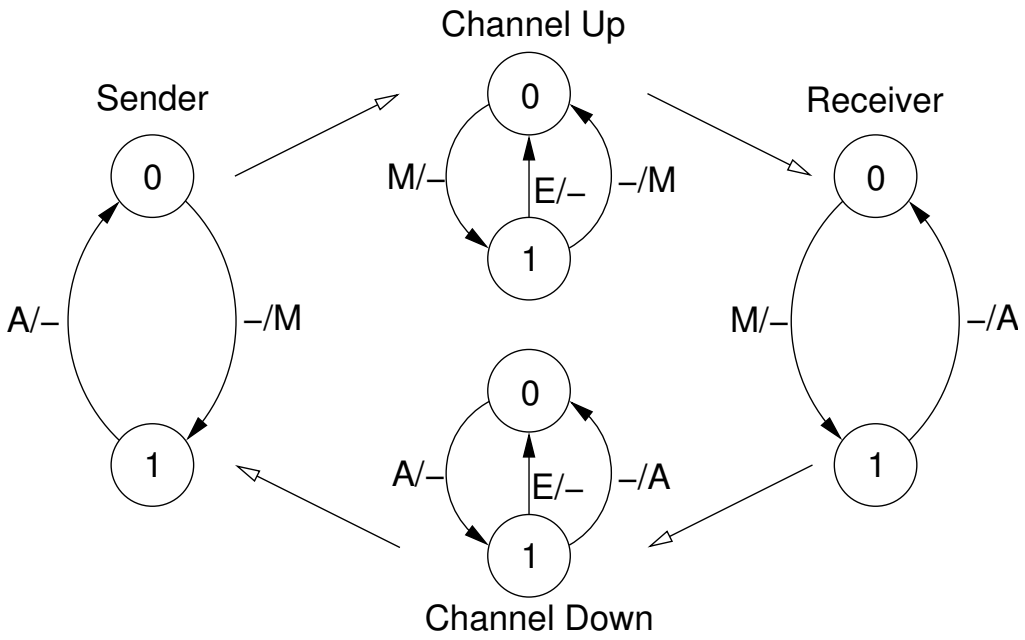
Graf dosažitelnosti – bez chyb



S	-	Cu	-	R	-	Cd	-	S
0	-	0	-	0	-	0	-	0
1	M	0	-	0	-	0	-	1
1	M	1	-	0	-	0	-	1
1	-	1	M	0	-	0	-	1
1	-	0	M	1	-	0	-	1
1	-	0	-	1	A	0	-	1
1	-	0	-	0	A	1	-	1
1	-	0	-	0	-	1	A	1
1	-	0	-	0	-	0	A	1
0	-	0	-	0	-	0	-	0

Jednoduchý potvrzovací protokol

Graf dosažitelnosti – s možností chyby

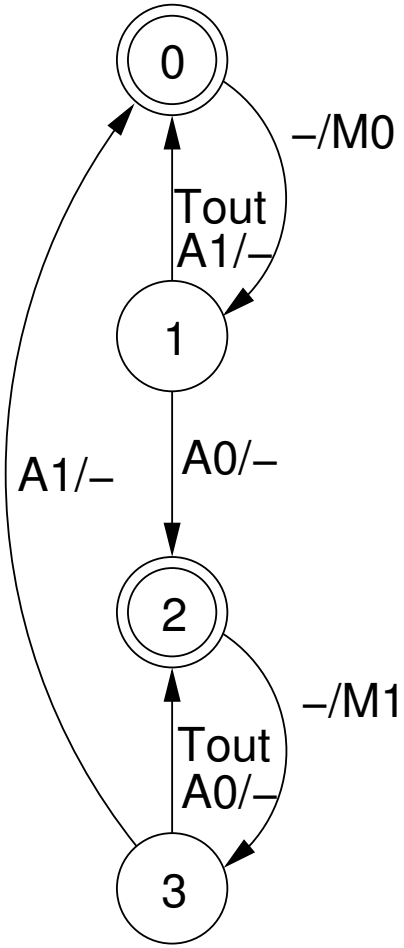


S	-	Cu	-	R	-	Cd	-	S
0	-	0	-	0	-	0	-	0
1	M	0	-	0	-	0	-	1
1	M	1	-	0	-	0	-	1
1	-	1	M	0	-	0	-	1
1	-	0	M	1	-	0	-	1
1	-	0	-	1	A	0	-	1
1	-	0	-	0	A	1	-	1
1	-	0	-	0	-	1	E	1
1	-	0	-	0	-	0	-	1
1	-	0	-	0	-	0	-	1

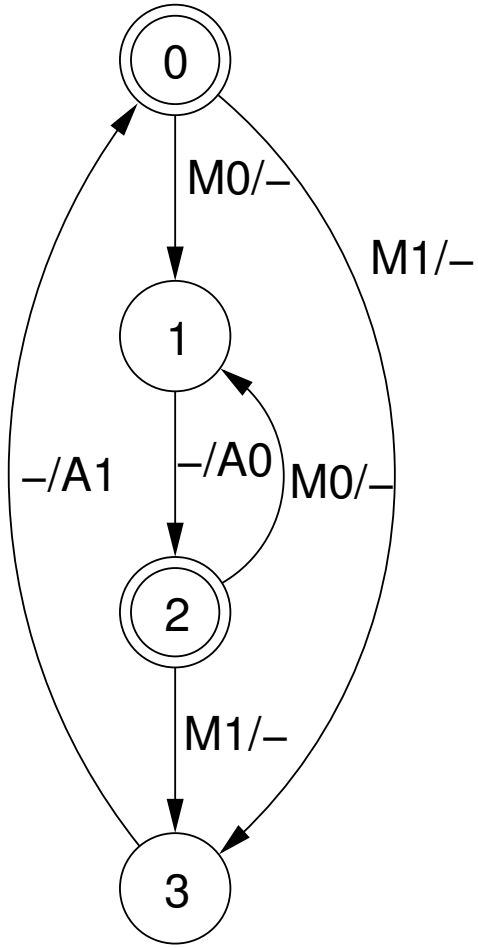
ztráta paketu $\Rightarrow$ nekonzistentní stav systému

Střídavé potvrzování
S kanálem tvořícím chyby

Sender

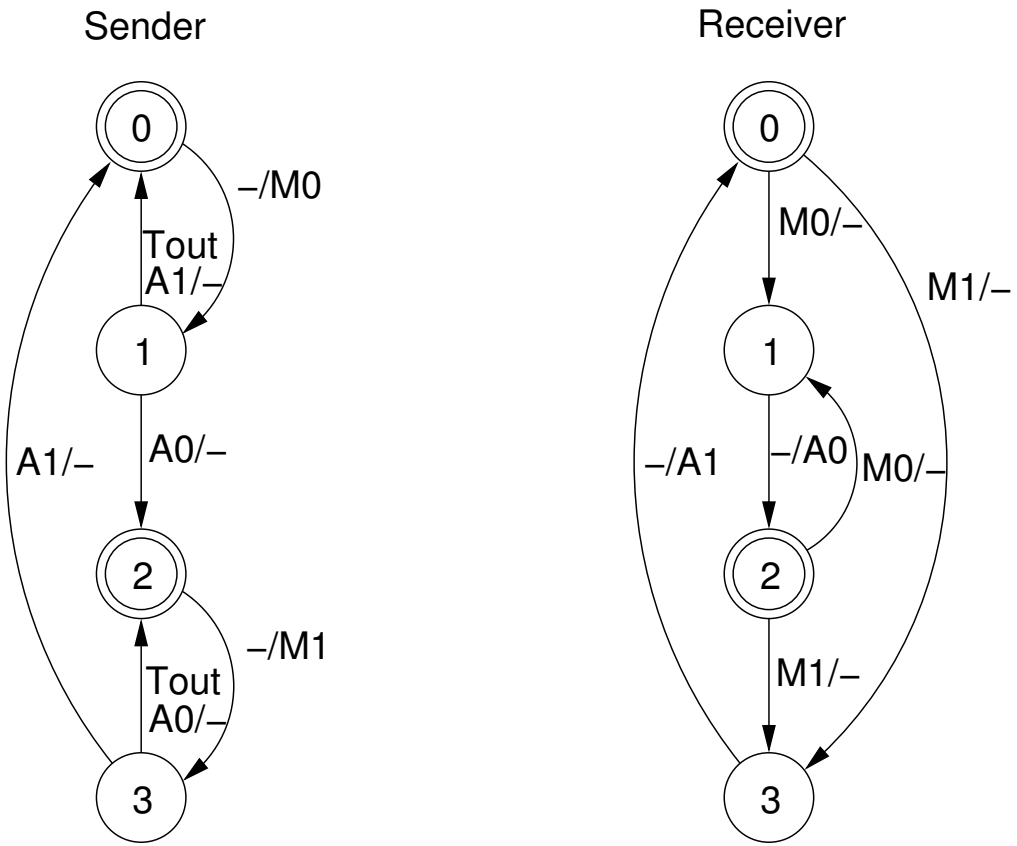


Receiver



Střídavé potvrzování

Graf dosažitelnosti – s možností chyby

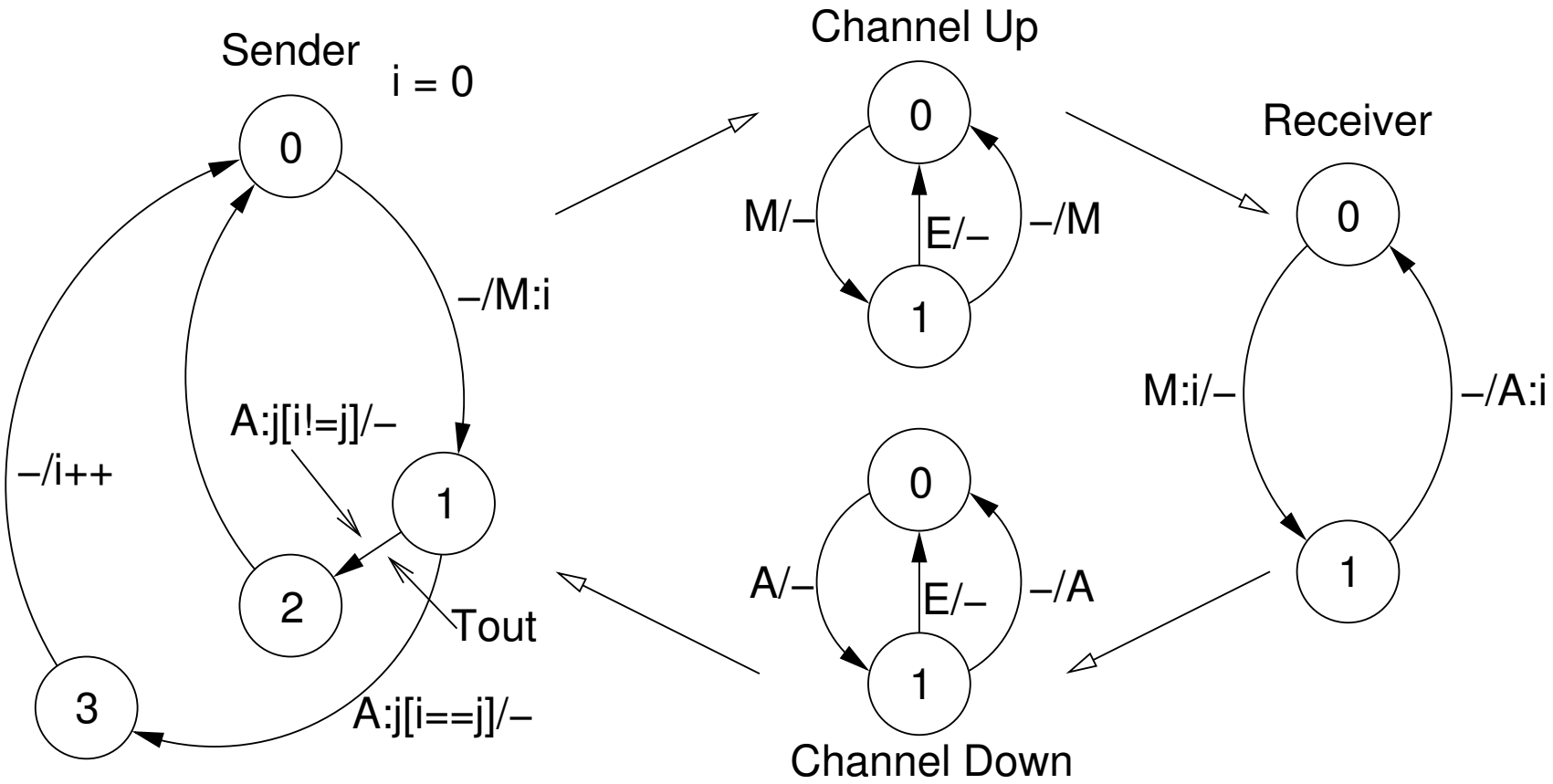


S	-	Cu	-	R	-	Cd	-	S
0	-	0	-	0	-	0	-	0
1	M	0	-	0	-	0	-	1
1	M	1	-	0	-	0	-	1
1	-	1	M	0	-	0	-	1
1	-	0	M	1	-	0	-	1
1	-	0	-	1	A	0	-	1
1	-	0	-	2	A	1	-	1
1	-	0	-	2	-	1	E	1
1	-	0	-	2	-	0	-	1
				Timeout				
0	-	0	-	2	-	0	-	0

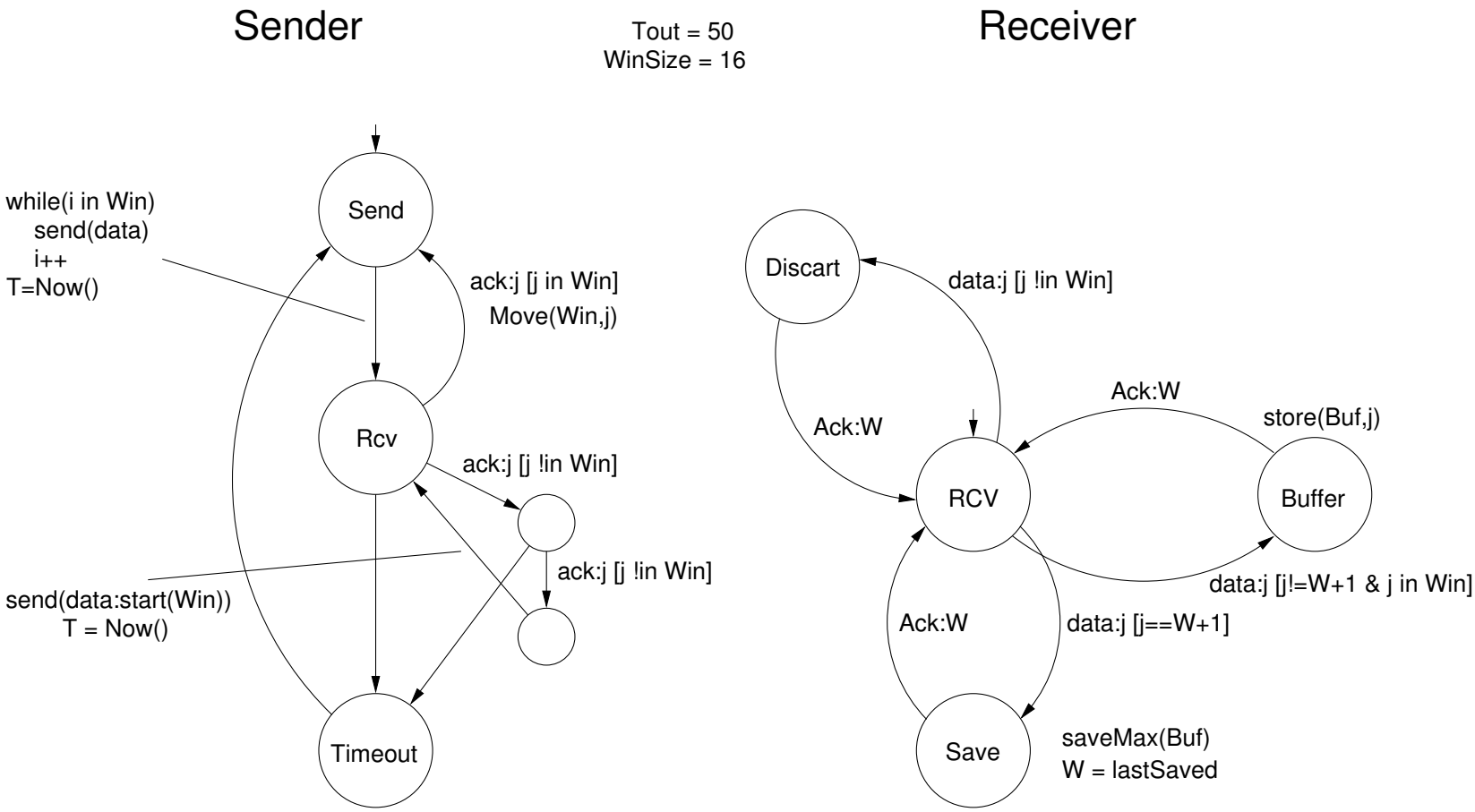
ztráta paketu $\Rightarrow$ timeout $\Rightarrow$ opakování přenosu

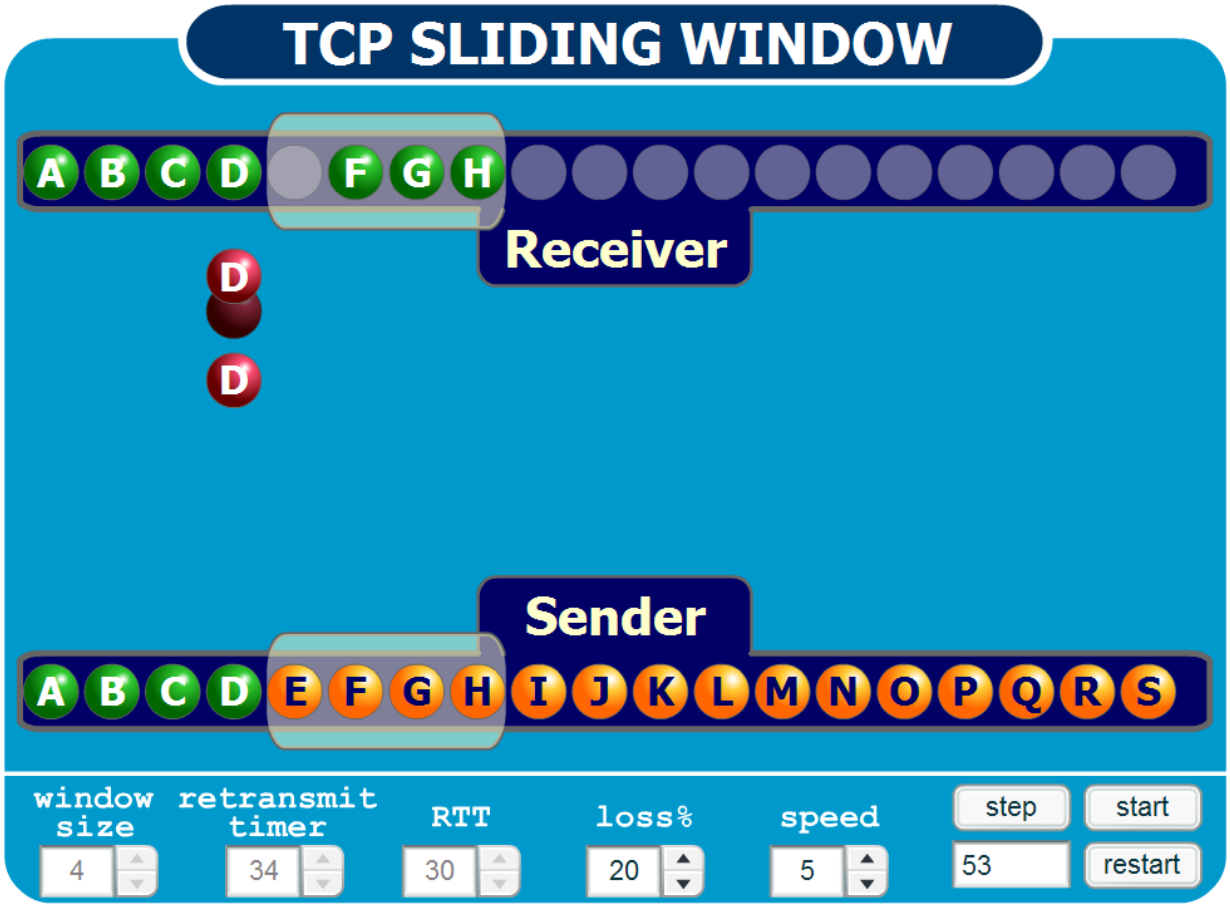
Střídavé potvrzování

Jiné zobrazení



- Číslování paketů (M:i – i je číslo paketu, 0 na začátku)
- Kontrola čísla potvrzení (A:j – j je číslo potvrzení)

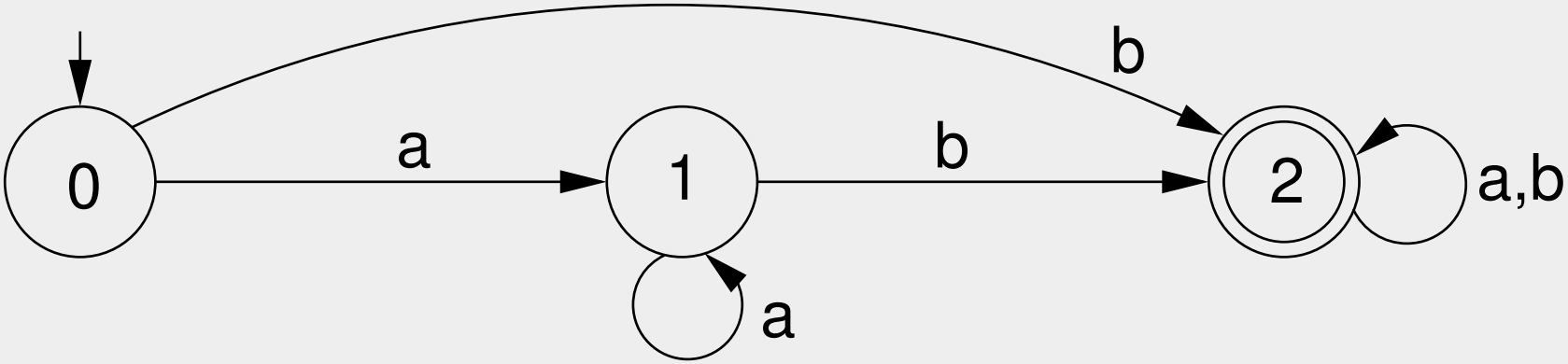

Sender
Tout = 50
WinSize = 16
Receiver
while(i in Win)
send(data)
i++
T=Now()
Send
ack:j [j in Win]
Move(Win,j)
Rcv
ack:j [j !in Win]
ack:j [j !in Win]
send(data:start(Win))
T = Now()
Timeout
Discart
data:j [j !in Win]
Ack:W
Ack:W
RCV
store(Buf,j)
Buffer
data:j [j!=W+1 & j in Win]
Ack:W
data:j [j==W+1]
Save
saveMax(Buf)
W = lastSaved



Sliding window demo

Následuje

Obrázek




```
int c = state = 0;
while((c = readChar())!=-1){
  switch(state){
    case 0:
      switch(c){
        case 'a': state = 1; break;
        case 'b': state = 2; break;
        default: error("Unknown char");
      } break;
    case 1:
      switch(c){
        case 'a': state = 1; break;
        case 'b': state = 2; break;
        default: error("Unknown char");
      } break;
    case 2:
      switch(c){
        case 'a': state = 2; break;
        case 'b': state = 2; break;
        default: error("Unknown char");
      } break;
  }
}
if(state == 2) print("OK");
else print("KO");
```

Switch

- switch (stavová proměnná) switch (událost)
- id stavu = číslo ve stavové proměnné

Procesy

- blokující/neblokující přijímání zpráv
- volání metod

Další

- Google
- FSM in games

Otázky ?

Ondřej Votava
votavon1@fel.cvut.cz