# Comunicación entre tres entidades con enlaces semidúplex.

Algoritmo a comprobar

## Problema a resolver

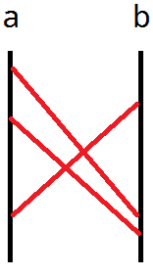
Formular un algoritmo que resuelva la comunicación entre tres entidades interconectadas con enlaces semidúplex (*half-duplex*).

Se asume que, en caso de colisión (que dos nodos intenten transmitir simultáneamente por el enlace que los conecta), los mensajes de ambos nodos se perderán. Las entidades no necesariamente detectan las colisiones.

El algoritmo propuesto considera que cualquier nodo puede recibir en cualquier momento un impulso espontáneo que le ordene enviar información a uno u otro vecino[1]. Dado que la letra del problema no especifica que la información se deba enviar directamente del nodo que la genera a su destinatario, se asume que es válido "rutear" el mensaje por un tercer nodo (por ejemplo que ***b*** actúe como intermediario en el envío de un mensaje de ***a*** hacia ***c***).

## Restricción empleada en el algoritmo

El algoritmo presentado asume que **una colisión afecta solo a un mensaje de cada nodo** participante. En otras palabras, se descarta que se den situaciones como las del siguiente diagrama en las que un único mensaje de un nodo interfiera con dos o más mensajes:




Esto permite asumir que si un nodo ***a*** envía *n* mensajes por un enlace y su vecino ***b*** le envía *n+1*, entonces los *n* mensajes de ***a*** colisionarán con los primeros ***b*** dejando el enlace "despejado" para que el último mensaje de ***b*** pueda ser entregado sin riesgo de colisión.

Nótese que esta es una restricción más débil que la de ser capaz de detectar colisiones: como se mostró en la clase todo sistema que detecte colisiones puede adaptarse para cumplir esta propiedad[2] mientras que lo opuesto es presumiblemente falso[3].

---

1 Dado que los nodos no conocen la identidad de sus vecinos (ni necesariamente poseen identificadores siquiera), en realidad el contenido de un impulso espontáneo sería en todo caso "tal información, a enviar al nodo que esté por tal enlace" (por la propiedad de *orientación local* las entidades sí pueden distinguir entre nodos).

2 Si se detectan colisiones, los nodos pueden esperar para enviar su segundo mensaje hasta o bien detectar la colisión (si se produjera) o bien hasta recibir un ACK (indicando que la colisión no se produjo).

3 Intuitivamente, la propiedad no debería ser suficiente para lograr detectar colisiones aunque no cuento con una prueba que me permita asegurarlo.

# Concepto del algoritmo

La comunicación entre las tres entidades sería trivial de no ser porque los enlaces utilizados provocan la pérdida de mensajes en caso de darse colisiones. Una idea intuitiva para resolver esto es tratar de evitar las colisiones por completo, para lo cual sería necesario un protocolo que impida que dos vecinos intenten comunicarse a la vez por el mismo enlace.

Una manera simple de lograr esto en una red de tres nodos interconectadas fijar un sentido para la transmisión y usar los tres enlaces semidúplex como si fueran enlaces unidireccionales en un anillo dirigido. Esto obligaría a tener que si un nodo quiere enviar información al vecino del cual *recibe* información, deberá transmitirlo a través del tercer nodo.

Usando este concepto, se puede pensar en un algoritmo que resuelva el problema en dos fases:

1. Una primera fase en la que los nodos deciden qué sentido utilizar para la comunicación (¿“$a \to b \to c \to a$” o “$a \to c \to b \to a$”?).
2. Una segunda fase en la que se da la comunicación en sí y en la que se evitan las colisiones porque los nodos no intentan nunca enviar datos por el enlace por el que “escuchan”.

Mientras que la segunda fase es trivial de resolver, la primera se enfrenta al problema de los enlaces semidúplex: los nodos deben poder coordinarse pese a las colisiones. Para salvar este obstáculo se buscará una manera en la los nodos envíen mensajes de tal manera que siempre haya *n+1* mensajes en un sentido que permitan sobrepasar *n* mensajes en el sentido opuesto (factible si la red tiene la propiedad mencionada anteriormente).

## Secuencia de mensajes

En el protocolo se propone lograr la coordinación a través de un número limitado de mensajes con una semántica definida. Dichos mensajes serán:

- Mensajes **ESTE-SENTIDO**: cuando un nodo se activa por un primer impulso espontáneo *sin haber recibido mensajes previamente*, enviará mensajes a sus vecinos sugiriendo un sentido para la formación del anillo dirigido. El sentido propuesto por un **ESTE-SENTIDO** es el mismo por el que vino el mensaje (es decir, que en la fase 2 el remitente ***x*** del mensaje sea quien transmita al nodo ***y*** que lo recibió, que ***y*** transmita al nodo restante ***z*** y que ***z*** cierre el ciclo transmitiéndole a ***x***).

  Por motivos que se explicarán más adelante, cada nodo podrá enviar un máximo de tres mensajes ESTE-SENTIDO: dos a un vecino y uno al otro.

- Mensajes **PIDO-CONFIRMAR**: Si un nodo ***y*** recibe un mensaje ESTE-SENTIDO de un nodo ***x***, procederá a responderle un mensaje **PIDO-CONFIRMAR** mediante el cual pedirá que ***x*** le confirme que está de acuerdo en seguir el sentido propuesto ($x \to y \to z$).

  Pedir esta confirmación es necesario ya que ***x*** pudo haberle enviado también un mensaje de ESTE-SENTIDO al nodo ***z***, proponiendo el sentido opuesto ($x \to z \to y$). Al pedir confirmación, el nodo ***x*** podrá definir cuál de las dos propuestas valida, respondiendo un CONFIRMADO únicamente al primero nodo que le pida confirmación.

- Mensajes **CONFIRMADO**: Un nodo ***x*** que previamente envió mensajes ESTE-SENTIDO responderá **CONFIRMADO** al primer nodo ***y*** que le pida confirmación. Una vez que ***x*** haya emitido un mensaje de este tipo hará caso omiso a cualquier mensaje de menor jerarquía (ESTE-SENTIDO o PIDO-CONFIRMAR).

- Mensajes **SENTIDO-DECIDIDO**: Cuando ***y*** reciba una respuesta CONFIRMADO de ***x*** sabrá que ambos nodos están de acuerdo en el sentido a utilizar, por lo que solo falta obligar al tercer nodo ***z*** a que también siga el rumbo acordado notificándolo con el mensaje **SENTIDO-DECIDIDO**. Una vez enviado este mensaje, ***y*** ignorará mensajes de menor jerarquía (ESTE-SENTIDO, PIDO-CONFIRMAR o un segundo CONFIRMADO en caso de que ***y*** hubiera recibido mensajes ESTE-SENTIDO de ambos vecinos y hubiera trasnsmitido exitosamente un PIDO-CONFIRMAR a los dos).

  Los mensajes **SENTIDO-DECIDIDO** requieren siempre un confirmador (el ***x*** que envió el CONFIRMADO) y un anunciador (el ***y*** que envía el SENTIDO-DECIDIDO), lo cual determina un sentido a tomar $x \to y \to z$. Podría existir otro par de nodos confirmador-anunciador que determinara otro mensaje **SENTIDO-DECIDIDO** pero este **siempre será en un sentido coherente** con $x \to y \to z$: puede que el par confirmador-anunciador sean $y \to z$ o $z \to x$ pero nunca $x \to z$, $y \to$ x o $z \to y$

  - Si se diera $x \to z$ entonces ***x*** debería haber enviado un CONFIRMADO tanto a ***y*** como a ***z***, lo cual es imposible ya que los nodos solo confirman el primer PIDO-CONFIRMAR que reciben.

  - Si se diera $y \to x$ entonces ***x*** e ***y*** se habrían confirmado el uno al otro, lo cual a su vez requeriría que se hubieran pedido confirmación el uno al otro y, en última instancia, que se hubieran enviado mensajes ESTE-SENTIDO el uno al otro. Según lo establecido, los mensajes ESTE-SENTIDO únicamente se envían *antes de recibir algún mensaje* por lo que queda descartado que ***x*** recibiera el ESTE-SENTIDO mensaje de ***y*** antes de enviar el propio o viceversa. Sería forzoso, por tanto, que ambos hubiesen enviado los mensajes simultáneamente lo cual habría resultado en colisión. Se concluye así que recibir exitosamente mensajes idénticos como en este caso es imposible, descartando la posibilidad.

  - Si se diera $z \to y$ entonces el nodo ***y*** tendría que haber reaccionado tanto al mensaje de CONFIRMADO de ***x*** como al de ***z***, lo cual va en contra del comportamiento mencionado en el que se ignoran los siguientes mensajes CONFIRMADO recibidos después del que causa el mensaje de SENTIDO-DECIDIDO.

- Una vez que un nodo ***z*** reciba un mensaje de SENTIDO-DECIDIDO sabrá que los otros dos nodos (el confirmador ***x*** y el anunciador ***z***) están de acuerdo en el sentido a seguir y que se comprometerán en no transmitir en el sentido contrario para evitar colisiones, por lo que ***z*** ya estaría en condiciones de transmitir datos libremente a ***x*** (y, mediante ruteo por ***x***, a ***y***). Sin embargo, estos otros dos nodos no necesariamente saben que ***z*** ya está enterado y que también se comprometerá a no transmitir en la dirección contraria.

  Esto se debe a que, considerando el conocimiento local de cada nodo:

- ***y*** todavía no puede transmitir hacia ***z*** sin temer colisiones ya que sería posible que ***z*** no hubiera adoptado todavía el sentido e intentara enviar algún mensaje “a contramano”.
- ***x***, mientras tanto, todavía no tiene constancia de que puede transmitirle a ***y*** ya que, desde su punto de vista, existiría la posibilidad de que ***y*** hubiera recibido confirmación del nodo ***z*** primero y que hubiera un mensaje SENTIDO-DECIDIDO en camino desde ***y***.

Para solucionar esto, ***z*** deberá re-enviar el **SENTIDO-DECIDIDO** a ***x*** y este a ***y*** para lograr que el conocimiento común (que todos sepan el sentido de la comunicación y estén además concientes de que los demás lo saben). Los mensajes SENTIDO-DECIDIDO se reenviarán hasta llegar a un nodo que ya haya enviado un mensaje de este tipo. Una vez que esto suceda, los nodos serán libres de transmitir los datos sin temor a colisiones siempre y cuando se siga el sentido pactado (usando a un nodo intermediario para mensajes “a contramano”).

## Repetir mensajes hasta asegurar su recepción

Para aplicar la secuencia de mensajes mencionada en la sección anterior se requiere una manera de lograr que los mensajes puedan ser transmitidos aun ante posibles colisiones. Para lograr esto se utilizarán tres herramientas:

- La propiedad que se le asume a la red de que 1 mensaje en un sentido interfiere a lo sumo con 1 mensaje en el sentido opuesto.
- El comportamiento de los iniciadores de *intentar* enviar dos mensajes ESTE-SENTIDO a un vecino y dos mensajes al otro.
- Que la red se compone de tres nodos interconectados.

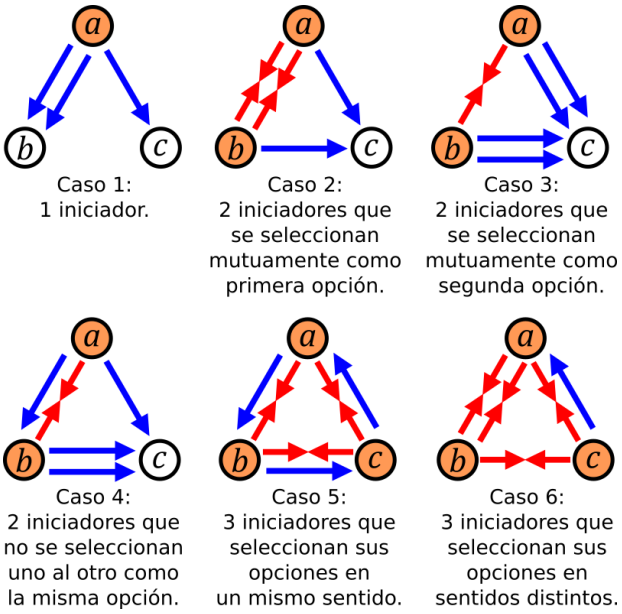
El primer punto permite asumir que si se conoce que un vecino intentará transmitir a lo sumo *n* mensajes, bastará con transmitir *n+1* mensajes para asegurar que recibirá al menos una vez el mensaje propio.

Combinando el primer punto con el segundo, se deduce que dado un par de nodos ***x*** e ***y*** se pueden dar las siguientes posibilidades al inicio del algoritmo:

1. Si solo uno de los nodos recibiera el impulso espontáneo, entonces no tendría inconvenientes en enviarle mensajes al otro.
2. Si ambos nodos reciben impulsos espontáneos antes de recibir algún mensaje (y, por tanto, ambos envían mensajes ESTE-SENTIDO):
   *a)* Cada nodo envía un mensaje al otro (y dos mensajes al tercero). Los dos mensajes colisionan y se pierden.
   *b)* Cada nodo envía dos mensajes al otro. Si lo hicieran en forma sincronizada, cada par de mensajes colisionaría, perdiéndose los cuatro mensajes (en dos colisiones).
   *c)* Si uno de los nodos envía un mensaje y el otro dos mensajes, se perderá el primer par de mensajes pero el mensaje repetido sí llegará con éxito.

Se deduce entonces que para asegurar que llegue algún mensaje ESTE-SENTIDO que permita iniciar

el resto del protocolo hace falta o bien que un nodo que solo reciba mensajes sin enviar ninguno (caso 1) o que un nodo mande dos mensajes a uno que solo le envía un mensaje (caso 2c). Afortunadamente, todas las configuraciones iniciales para una red de tres nodos interconectados permiten que se de al menos uno de estos casos:




De esta forma se concluye que al menos un mensajes ESTE-SENTIDO llega exitosamente, a partir del cual el algoritmo puede continuar.

El siguiente paso es asegurar que sea posible enviar un mensaje PIDO-CONFIRMAR desde un nodo ***y*** en respuesta a un ESTE-SENTIDO de un nodo ***x***. Dado que ***x*** pudo haber enviado dos mensajes ESTE-SENTIDO hacia ***y***, es posible que un intento de envío de PIDO-CONFIRMAR iniciado por un primer mensaje de ***x*** hacia ***y*** colisiones con un segundo mensaje de ***x***. Para asegurar que el PIDO-CONFIRMAR llegue basta con enviarlo dos veces, de forma que aún si el primer envío colisiona con un mensaje de ***x***, el segundo llegue (a menos que ***x*** haya enviado más mensajes a ***y***, lo cual indicaría que está en una etapa más avanzada del protocolo y se le puede dar preferencia[4]).

Algo similar ocurre con los siguientes mensajes. Cuando un nodo ***x*** reciba un mensaje de PIDO-CONFIRMAR será posible que corresponda el primer o al segundo intento de envío del mismo, por lo que el mensaje CONFIRMADO también debe enviarse dos veces. Los mensajes de SENTIDO-DECIDIDO, mientras tanto, podrían tener que enfrentar colisiones con hasta dos otros mensajes (por ejemplo dos mensajes ESTE-SENTIDO diferidos que le intentara enviar el tercer nodo) por lo que hacen falta tres intentos para asegurar la recepción.

De esta forma (siempre y cuando el algoritmo propuesto o su razonamiento no contengan algún error) los tres nodos podrán coordinar una dirección para su comunicación.

4 Por ejemplo, ***x*** podría haber completado todos los pasos con ***z*** hasta llegar a enviar un SENTIDO-DECIDIDO a ***y***. En tal caso, resulta de poca importancia que no llegue el PIDO-CONFIRMAR y conviene darle paso a ***x***.

# Algoritmo

## Estados

Se emplean los siguientes estados:

- **IDLE**: Estado inicial de nodos que todavía no han recibido ni mensajes ni impulsos.
- **WAITING**: Nodos que enviaron un ESTE-SENTIDO y esperan respuestas.
- **ASKING**: Nodos que enviaron un PIDO-CONFIRMAR y esperan respuestas.
- **CONFIRMING**: Nodos que enviaron un CONFIRMADO y esperan un SENTIDO-DECIDIDO.
- **READY**: Nodos que enviaron un SENTIDO-DECIDIDO.
- **DONE**: Nodos que recibieron un SENTIDO-DECIDIDO.

## Pseudocódigo

**Nota:** en este pseudocódigo cuando un nodo tiene que enviar más de un mensaje (como cuando envía los 2+1 mensajes ESTE-SENTIDO) se envían en la misma acción atómica. Sería posible esperar entre los mensajes (mediante alarmas) para tener la posibilidad de, por ejemplo, recibir la respuesta al primer ESTE-SENTIDO antes de tener que enviar un segundo. Sin embargo, prefiero evitarlo por dos motivos: primeramente la claridad (el código con alarmas es notoriamente más largo y menos legible) y, en segundo lugar, que el uso de alarmas no debería ser esencial para que el algoritmo funcione.

Si ya se tiene información para enviar, un nodo puede dejar implícito el SENTIDO-DECIDIDO y simplemente enviar la información apropiada. Sin embargo, como hay que repetir tres veces los mensajes de SENTIDO-DECIDIDO para asegurar de que lleguen ya que puede no estar despejado aún el canal “a contramano”, cabría la posibilidad de que la información se recibiera erróneamente múltiples veces en caso de repetirla o que se perdiera en una colisión en caso de enviarla una sola vez. Esto se puede solucionar al enviar primero dos mensajes SENTIDO-DECIDIDO (que despejan las posibles colisiones) y recién luego la información cuando se tiene la certeza de que se recebirá correctamente.

```
if state.is_IDLE:
    sponaneously(info,link_num):
        info_for_send[link_num] = [info]
        send(ESTE_SENTIDO,link_1)
        send(ESTE_SENTIDO,link_2)
        send(ESTE_SENTIDO,link_1)
        become(WAITING)
    receiving(msg, link):
        if msg = ESTE-SENTIDO:
            send(PIDO_CONFIRMAR,link)
            send(PIDO_CONFIRMAR,link)
            become(ASKING)
        if msg = SENTIDO_DECIDIDO:
            in_link = link
            out_link = link_1 if link == link_2 else link2
            send(SENTIDO_DECIDIDO,out_link)
            send(SENTIDO_DECIDIDO,out_link)
            send(SENTIDO_DECIDIDO,out_link)
            info_for_send = []
            become(DONE)
        if msg = INFO:
            in_link = link
            out_link = link_1 if link == link_2 else link2
            info_for_send = []
            for data in INFO:
                if data.forward:
                    info_for_send += (data,not_forward)
                else:
                    process(data)
            send(SENTIDO_DECIDIDO,out_link)
            send(SENTIDO_DECIDIDO,out_link)
```

```
            if info_for_send.empty:
                send(SENTIDO_DECIDIDO,out_link)
            else:
                send(info_for_send,out_link)
                info_for_send = []
            become(DONE)
        else:
            ignore

if state.is_WAITING:
    spontaneously(info,link_num):
        info_for_send[link_num] += [info]
    receiving(msg, link):
        if msg = PIDO_CONFIRMAR:
            out_link = link
            in_link = link_1 if link == link2 else link2
            send(CONFIRMADO,out_link)
            send(CONFIRMADO,out_link)
            become(CONFIRMING)
        if msg = SENTIDO_DECIDIDO:
            in_link = link
            out_link = link_1 if link == link_2 else link2
            info_for_send = []
            for data in info_for_send[out_link]:
                info_for_send += (data,not_forward)
            for data in info_for_send[in_link]:
                info_for_send += (data,forward)
            send(SENTIDO_DECIDIDO,out_link)
            send(SENTIDO_DECIDIDO,out_link)
            send(info_for_send,out_link)
            info_for_send = []
```

```
            become(DONE)
        if msg = INFO:
            in_link = link
            out_link = link_1 if link == link_2 else link2
            info_for_send = []
            for data in info_for_send[out_link]:
                info_for_send += (data,not_forward)
            for data in info_for_send[in_link]:
                info_for_send += (data,forward)
            for data in INFO:
                if data.forward:
                    info_for_send += (data,not_forward)
                else:
                    process(data)
            send(SENTIDO_DECIDIDO,out_link)
            send(SENTIDO_DECIDIDO,out_link)
            send(info_for_send,out_link)
            info_for_send = []
            become(DONE)
        else:
            ignore

if state.is_ASKING:
    spontaneously(info,link_num):
        info_for_send[link_num] += [info]
    receiving(msg, link):
        if msg = PIDO_CONFIRMAR:
            if out_link is undefined:
                out_link = link
                in_link = link_1 if link == link2 else link2
                send(CONFIRMADO,out_link)
```

```
            send(CONFIRMADO,out_link)
        else:
            ignore
    if msg = CONFIRMADO:
        in_link = link
        out_link = link1 if link2 == link else link2
        send(SENTIDO_DECIDIDO,out_link)
        send(SENTIDO_DECIDIDO,out_link)
        send(SENTIDO_DECIDIDO,out_link)
        become(READY)
    if msg = SENTIDO_DECIDIDO:
        in_link = link
        out_link = link_1 if link == link_2 else link2
        info_for_send = []
        for data in info_for_send[out_link]:
            info_for_send += (data,not_forward)
        for data in info_for_send[in_link]:
            info_for_send += (data,forward)
        send(SENTIDO_DECIDIDO,out_link)
        send(SENTIDO_DECIDIDO,out_link)
        if info_for_send.empty:
            send(SENTIDO_DECIDIDO,out_link)
        else:
            send(info_for_send,out_link)
            info_for_send = []
        become(DONE)
    if msg = INFO:
        in_link = link
        out_link = link_1 if link == link_2 else link2
        info_for_send = []
        for data in info_for_send[out_link]:
```

```
                info_for_send += (data,not_forward)
            for data in info_for_send[in_link]:
                info_for_send += (data,forward)
            for data in INFO:
                if data.forward:
                    info_for_send += (data,not_forward)
                else:
                    process(data)
            send(SENTIDO_DECIDIDO,out_link)
            send(SENTIDO_DECIDIDO,out_link)
            send(info_for_send,out_link)
            info_for_send = []
            become(DONE)
        else:
            ignore

if state.is_CONFIRMING:
    spontaneously(info,link_num):
        info_for_send[link_num] += [info]
    receiving(msg, link):
        if msg = SENTIDO_DECIDIDO:
            in_link = link
            out_link = link_1 if link == link_2 else link2
            info_for_send = []
            for data in info_for_send[out_link]:
                info_for_send += (data,not_forward)
            for data in info_for_send[in_link]:
                info_for_send += (data,forward)
            send(SENTIDO_DECIDIDO,out_link)
            send(SENTIDO_DECIDIDO,out_link)
            send(info_for_send,out_link)
```

```
            info_for_send = []
            become(DONE)
        if msg = INFO:
            in_link = link
            out_link = link_1 if link == link_2 else link2
            info_for_send = []
            for data in info_for_send[out_link]:
                info_for_send += (data,not_forward)
            for data in info_for_send[in_link]:
                info_for_send += (data,forward)
            for data in INFO:
                if data.forward:
                    info_for_send += (data,not_forward)
                else:
                    process(data)
            send(SENTIDO_DECIDIDO,out_link)
            send(SENTIDO_DECIDIDO,out_link)
            send(info_for_send,out_link)
            info_for_send = []
            become(DONE)
        else:
            ignore

if state.is_READY:
    spontaneously(info,link_num):
        info_for_send[link_num] += [info]
    receiving(msg, link):
        if msg = SENTIDO_DECIDIDO:
            info_for_send = []
            for data in info_for_send[out_link]:
                info_for_send += (data,not_forward)
```

```
            for data in info_for_send[in_link]:
                info_for_send += (data,forward)
            send(SENTIDO_DECIDIDO,out_link)
            send(SENTIDO_DECIDIDO,out_link)
            if info_for_send.empty:
                send(SENTIDO_DECIDIDO,out_link)
            else:
                send(info_for_send,out_link)
                info_for_send = []
            become(DONE)
        if msg = INFO:
            info_for_send = []
            for data in info_for_send[out_link]:
                info_for_send += (data,not_forward)
            for data in info_for_send[in_link]:
                info_for_send += (data,forward)
            for data in INFO:
                if data.forward:
                    info_for_send += (data,not_forward)
                else:
                    process(data)
            send(SENTIDO_DECIDIDO,out_link)
            send(SENTIDO_DECIDIDO,out_link)
            send(info_for_send,out_link)
            info_for_send = []
            become(DONE)
        else:
            ignore


if state.is_DONE:
```

```
spontaneously(info,link_num):
    for data in INFO:
        if data.forward:
            info_for_send += (data,not_forward)
        else:
            process(data)
    if not info_for_send.empty:
        send(info_for_send,out_link)
        info_for_send = []
receiving(msg, link):
    if msg = INFO:
        for data in INFO:
            if data.forward:
                info_for_send += (data,not_forward)
            else:
                process(data)
        send(info_for_send,out_link)
        info_for_send = []
        become(DONE)
    else:
        ignore
```