User Stories

Endymion - Member of the IPS team

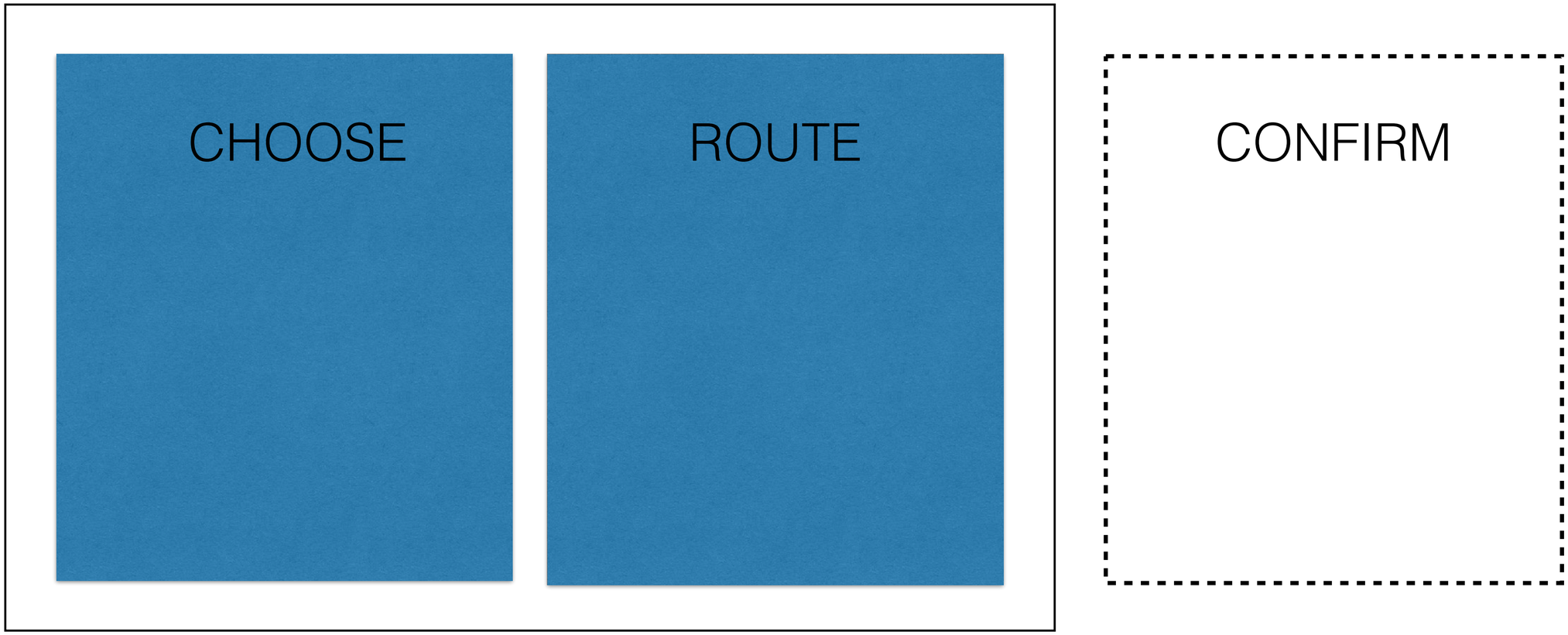
CREATE A NEW ROUTE

- open web router in browser
- select/ find sources and destinations that they are interested in
- having selected these, the UI displays these to them
- identify & select source that they would like to route from.
- UI displays valid potential destinations for route/s
- select a destination
- the UI displays to the user that a route is being created
- the UI displays to the user that a route has been created
- This can happen as many times as desired (including from the same source)

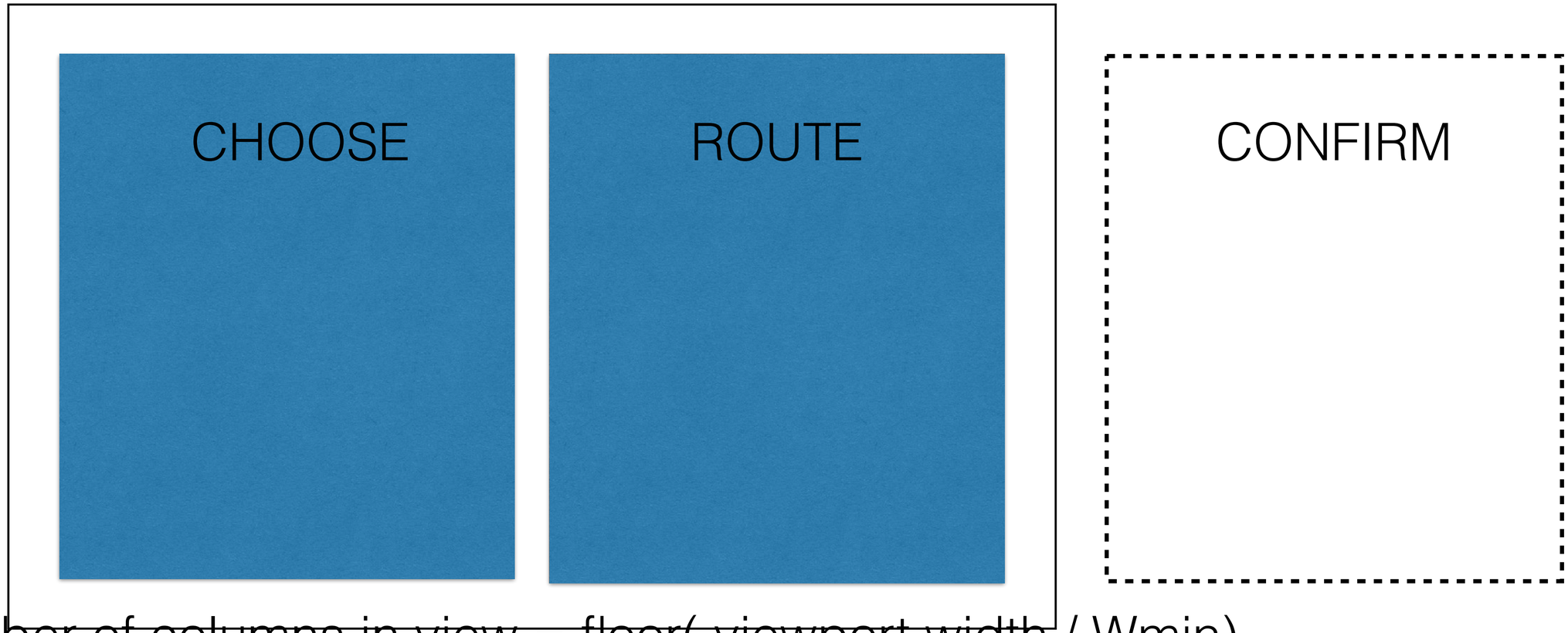
DELETE A ROUTE

- open web router in browser
- select/ find sources and destinations that they are interested in
- having selected these, the UI displays these to them
- identify & select the route that they would like to destroy using UI ‘destroy’ element.
- the UI displays to the user that a route is being destroyed
- the UI displays to the user that a route has been destroyed
- This can happen as many times as desired (including from the same source)

Overall Design

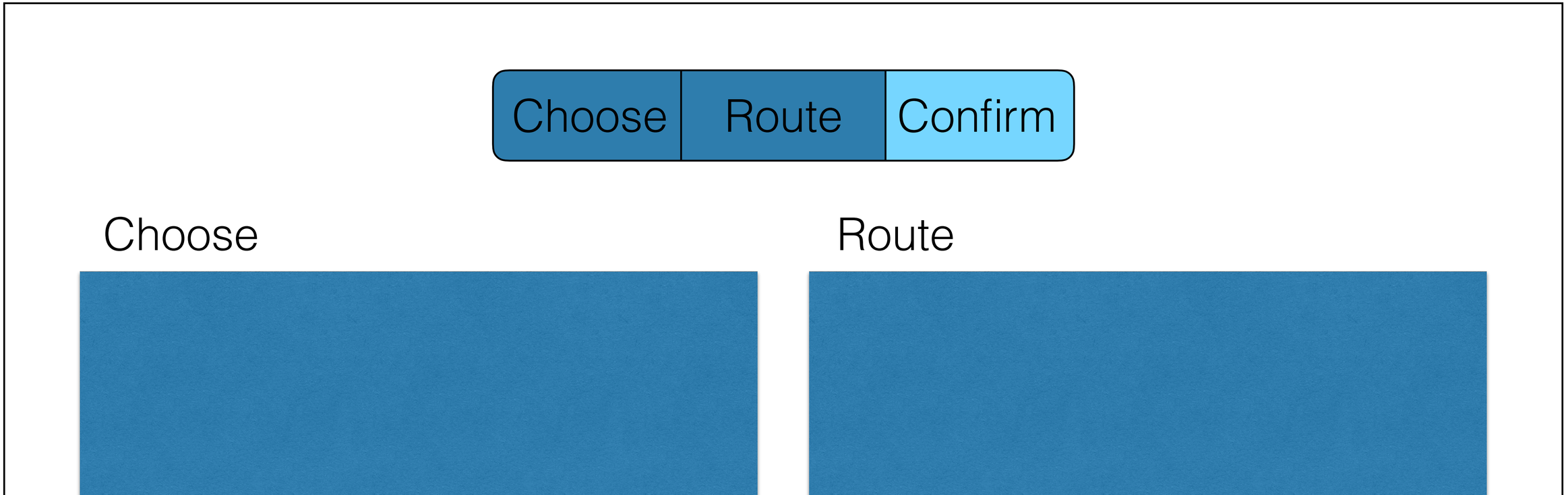


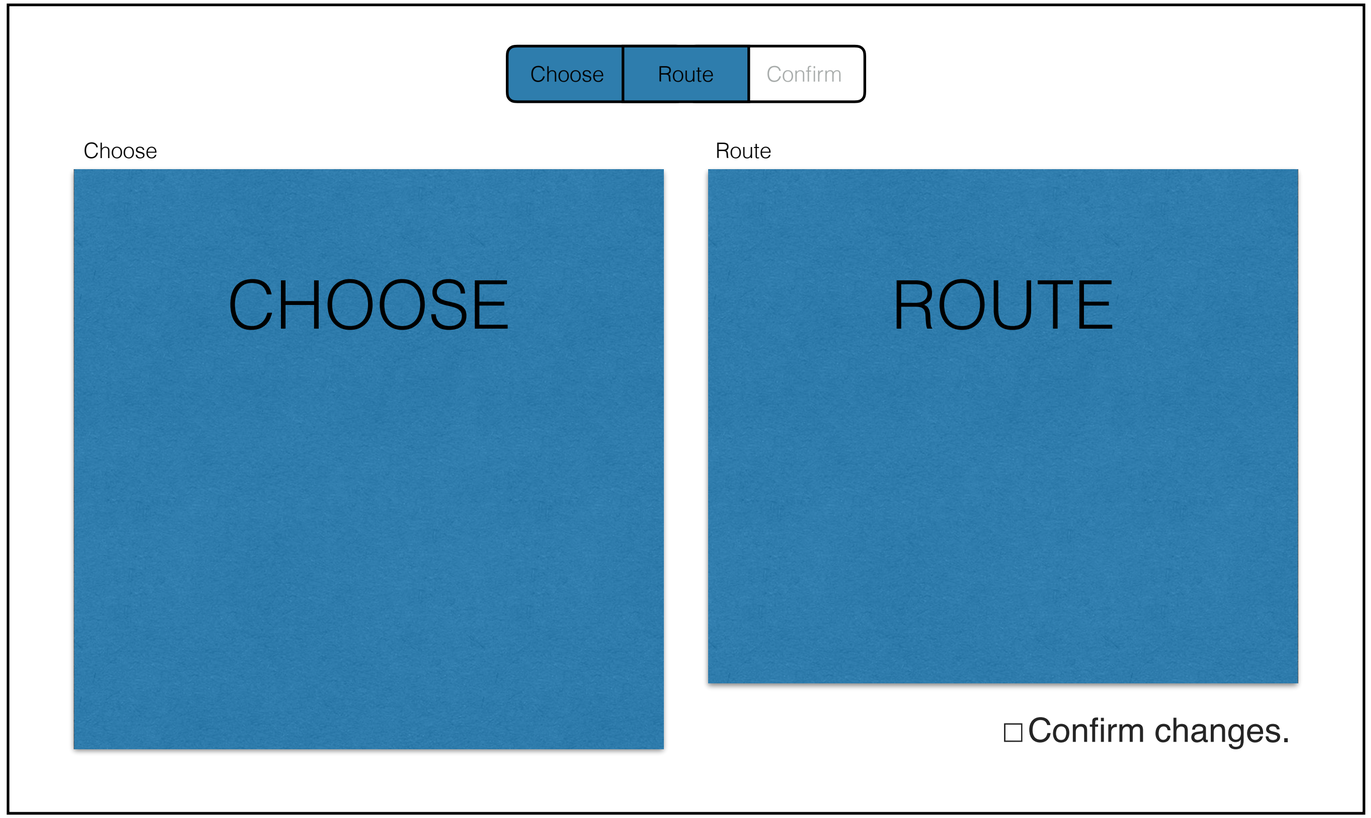
App is a multi-column view, with responsive layout altering the number of visible columns based on viewport width. All columns are the same width.
User journeys flow left to right: first the sources and destinations of interest are selected, then routes between these chosen routables are shown / edited. If route confirmation is enabled, this populates a confirmation list that shows the proposed changes and allows the user to accept them all at once or reject them one by one.



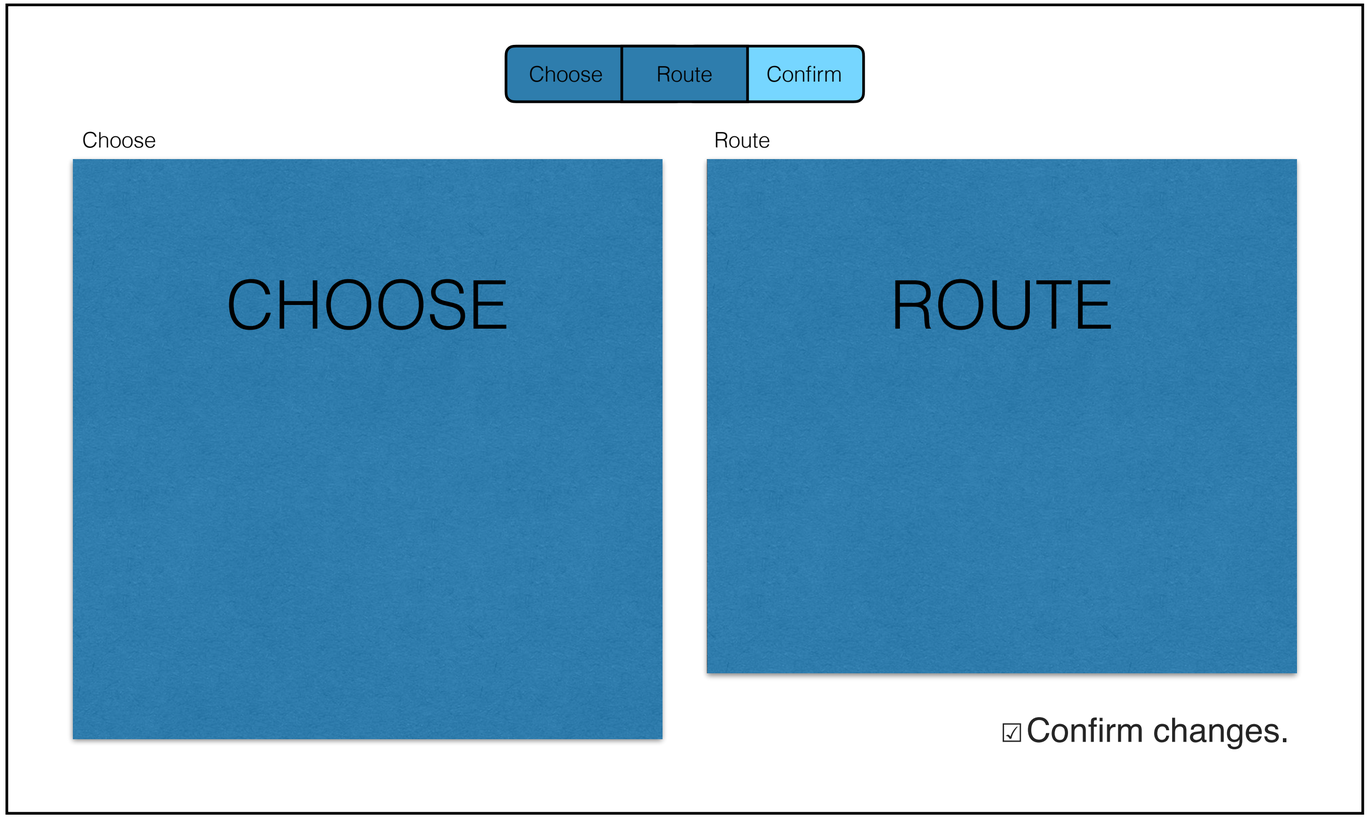
Number of columns in view = floor(viewport width / Wmin)
where Wmin is a minimum width to be determined that works well for all column types (640px?)
Actual column width (inc margin/padding) = min(viewport width / number of columns, Wmax)
where Wmax is a maximum width to be determined that satisfies the relationship Wmax <= 2 x Wmin. (Suggest trying Wmax = 2 x Wmin initially.)

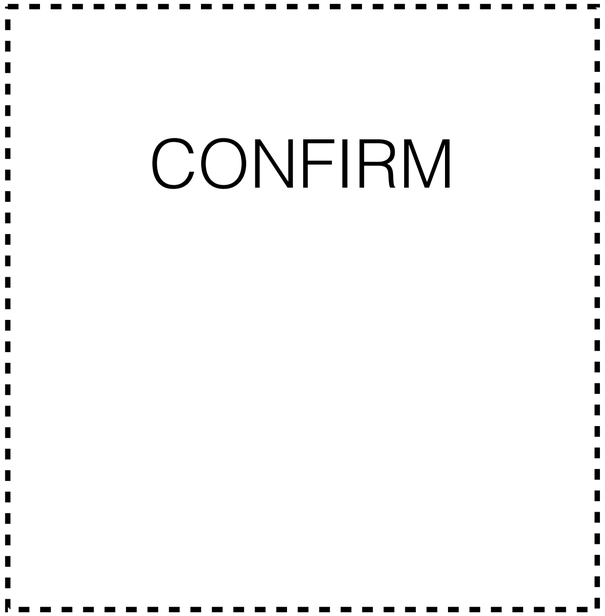
Column selection is based on a multi-button element, shown here for a two-column viewport in “confirm changes” mode. The buttons for the (two) visible columns are shown depressed. Clicking on the third button scrolls the viewport (animate 0.5s) to show that column plus the centre column. The button states update to reflect the change. The button for the centre column is always depressed, and clicking on it has no effect. Column headings clarify what each button corresponds to.



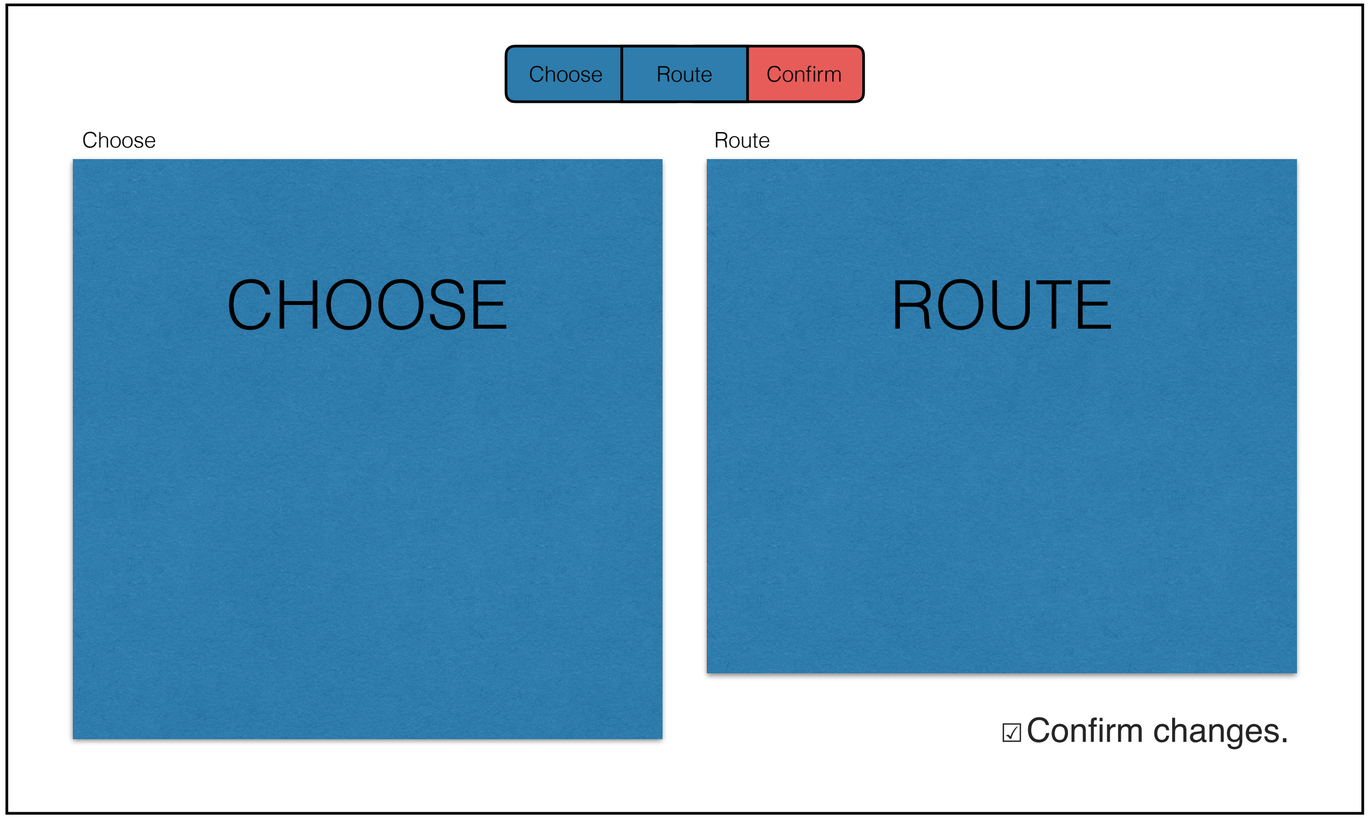


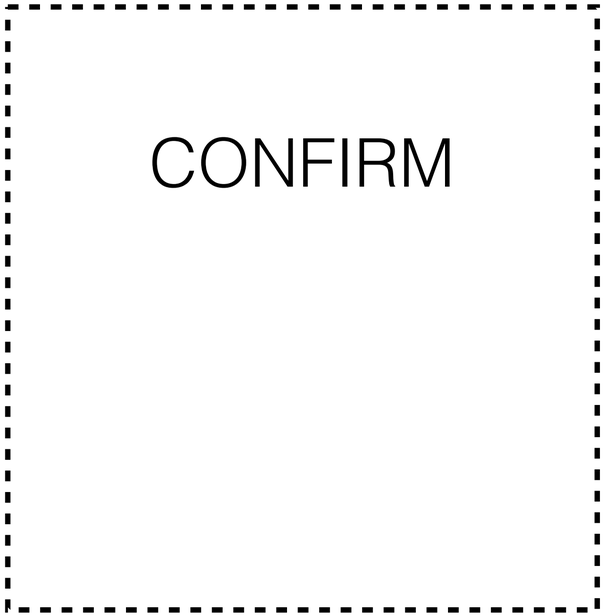
A “confirm changes” checkbox switches the UI between “instant apply” and “confirm changes” mode. When switching from “confirm changes” to “instant apply”, any unconfirmed changes are applied immediately as if the “take” button had been pressed.
If the “confirm” column is visible, the view scrolls left until it disappears. If the viewport width is such that the “confirm” column would be visible with the view scrolled fully left, the “confirm” column fades out (animate 0.25s) and then the other columns move to their positions dictated by the layout algorithm for a two-column view. If the column selection buttons are visible, the “confirm” button is shown disabled, and pressing it has no effect.



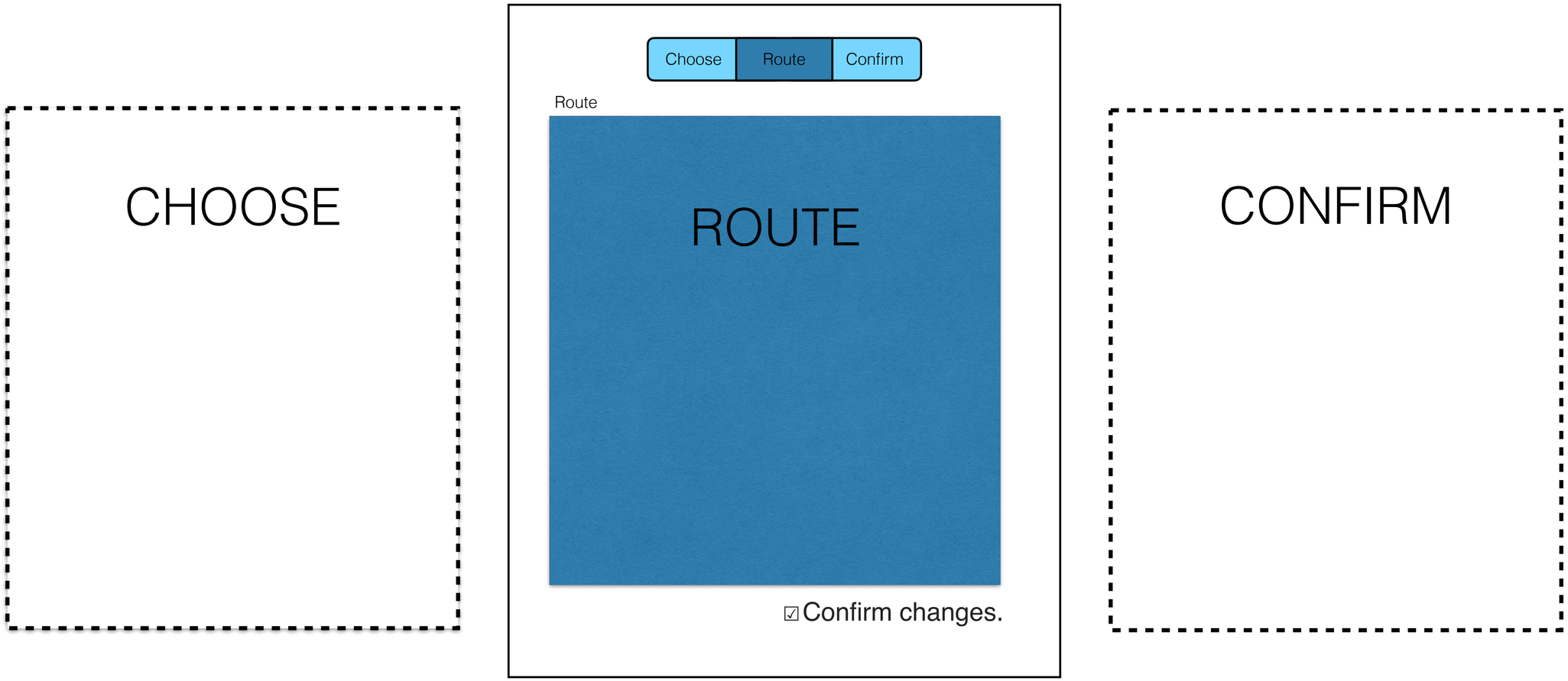


When switching from “instant apply” to “confirm changes” mode, the “confirm” column selection button is shown as enabled again. If the viewport width is wide enough to show all the columns at once, the other columns move and scale to their positions in the new layout (animate 0.25s) and then the “confirm” column fades in immediately (animate 0.25s). Otherwise it is not shown until the user makes it visibl (e.g. scrolls the viewport or makes it big enough for the column to be shown.)

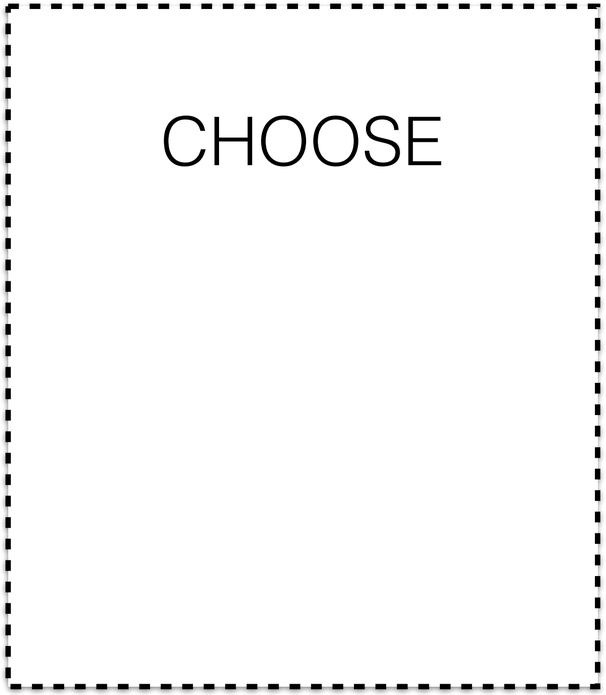


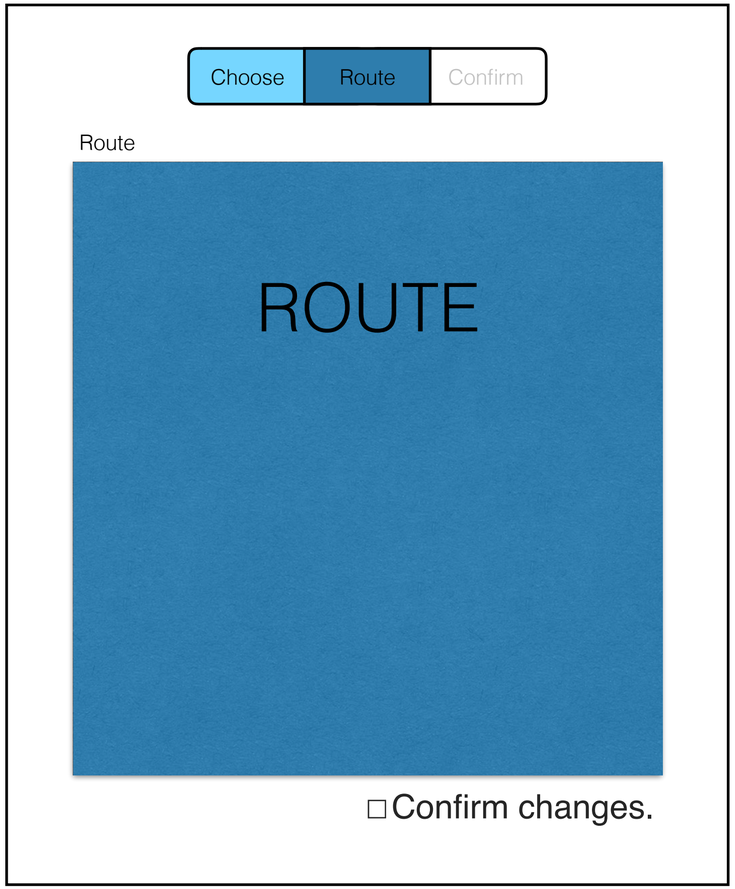


In “confirm changes” mode, every time the user adds a route or unroute operation to the confirmation list, then if the “confirm” column is not visible, the “confirm” button colour changes to red. It returns to its default colour the next time the “confirm” column is shown in the viewport.

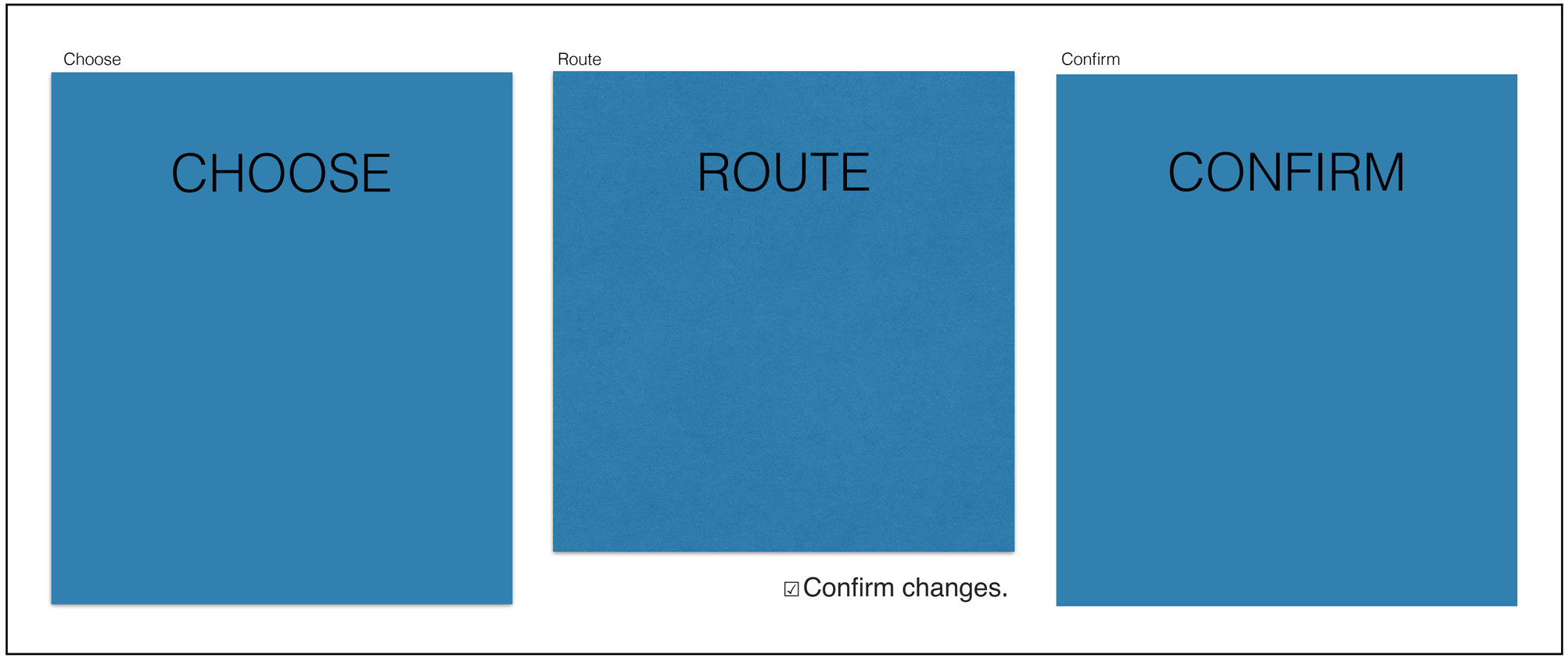


In single-column viewport mode (shown here in “confirm changes” mode), just the button representing the currently visible column is shown depressed. Pressing a different button scrolls that column into view and updates the buttons accordingly.





In “instant-apply” mode, just the button representing the currently visible column is shown depressed. Pressing a different button scrolls that column into view and updates the buttons accordingly.



If the viewport is wide enough to show all the columns at once, the buttons are not shown at all. If the viewport width changes while the app is running, such that the buttons should appear or disappear, this should be shown as the columns scrolling down to make room (animate 0.25s) followed by the buttons fading in (animate 0.25s) when they appear, and the reverse when they disappear.

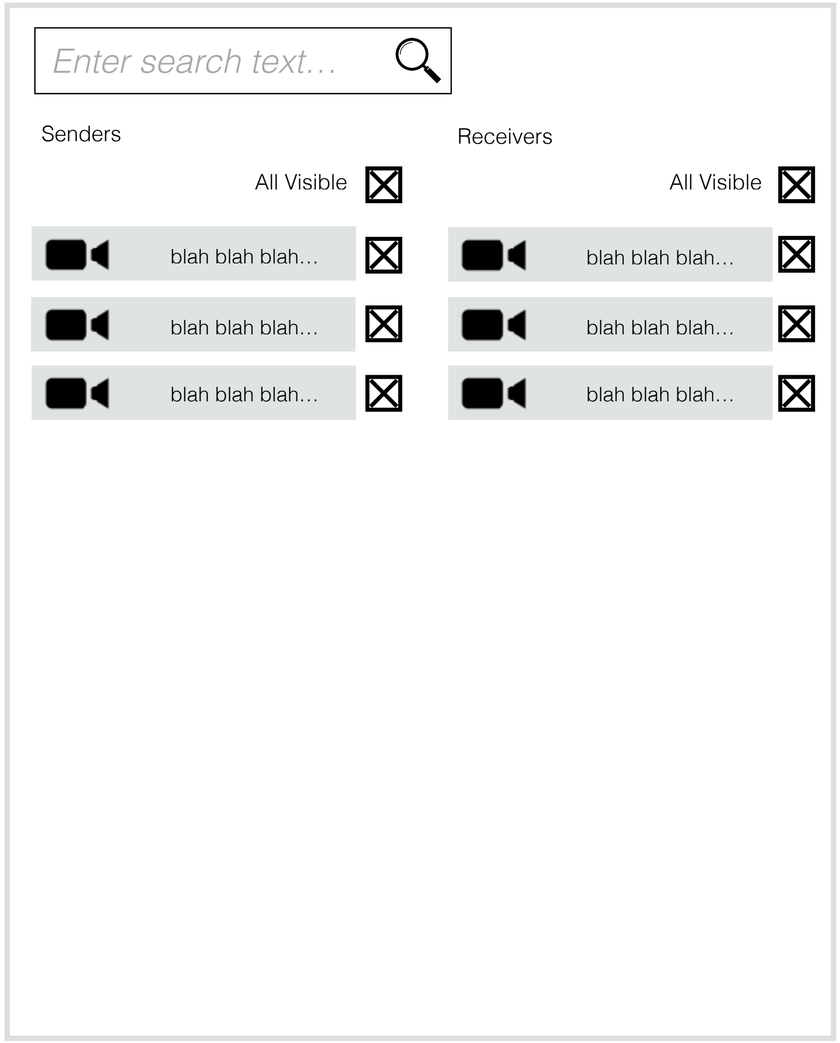
“Choose” View

All | My Lists | Devices

Insert Magical UX Here

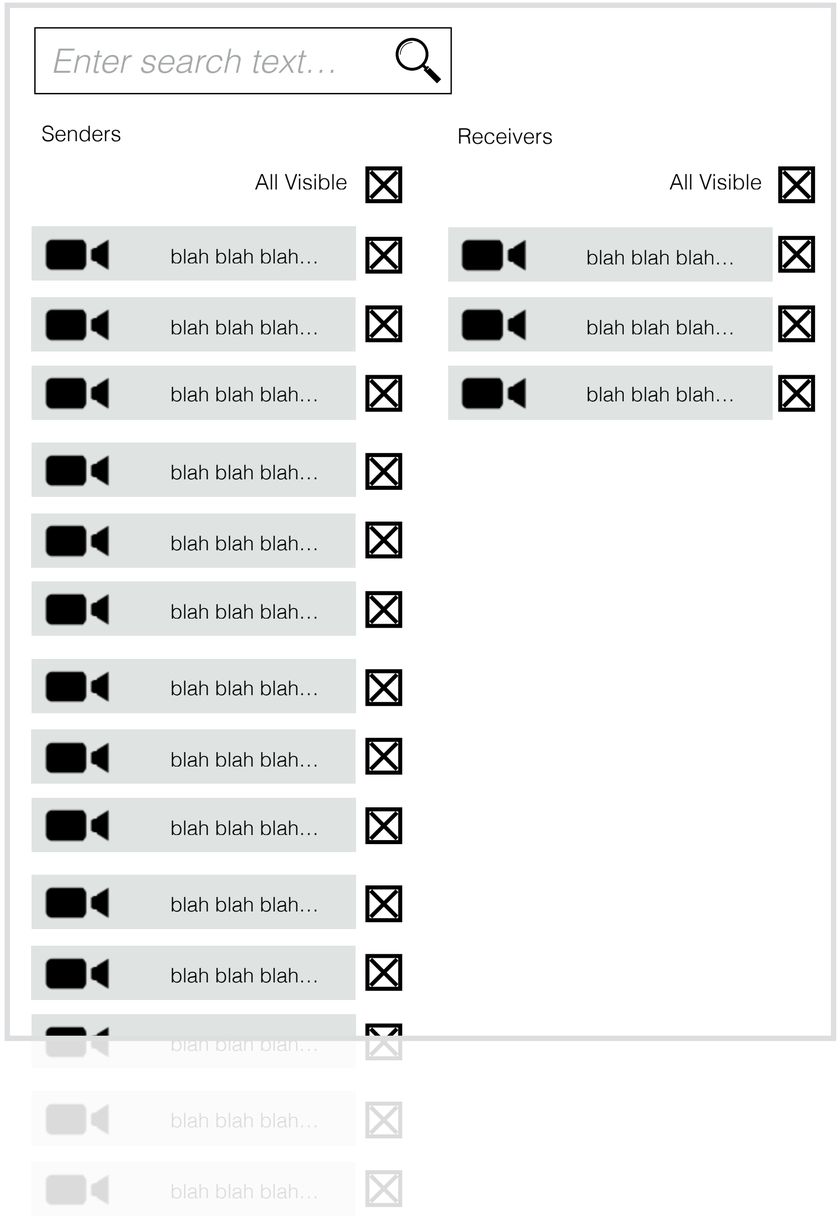
The long-term vision is that the “choose” view will be a tabbed interface that offers the user multiple ways to choose the routables of interest to them, including custom lists that they have saved earlier, templated groups fetched from the system configuration APIs, device-based groups, etc. In the MVP, a single way to choose routables is proposed.

Choose

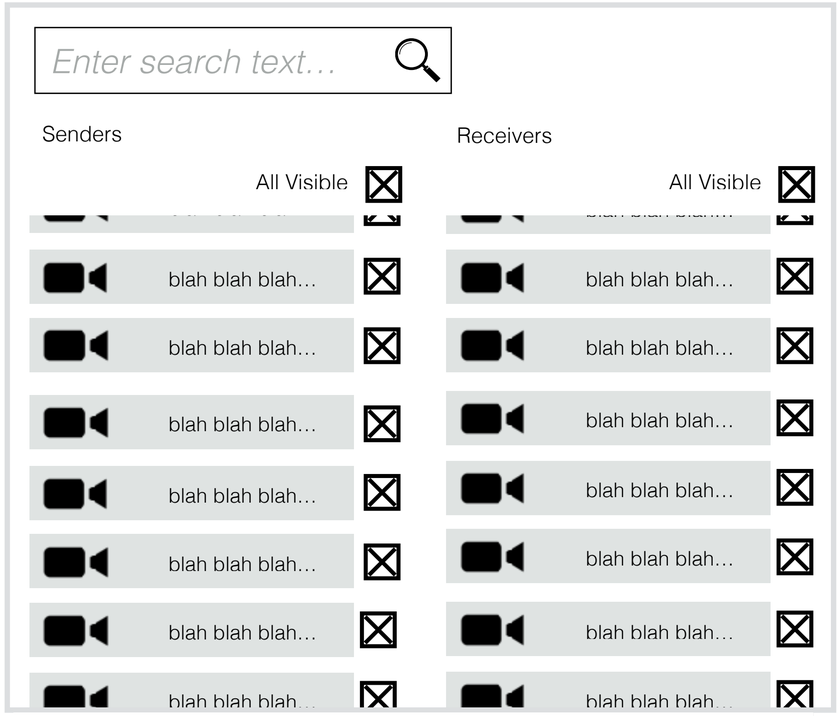


We lose the tabs for now, and just implement a simple dual list view with a search box and select/deselect visible buttons. When the view appears, the search box is empty, displaying a greyed italicised placeholder text. All senders and receivers are shown. Against each routable, a checkbox is shown. Initially, all routables are visible and all checkboxes are checked. If the “Route” view is on-screen, all senders and receivers are shown in it.

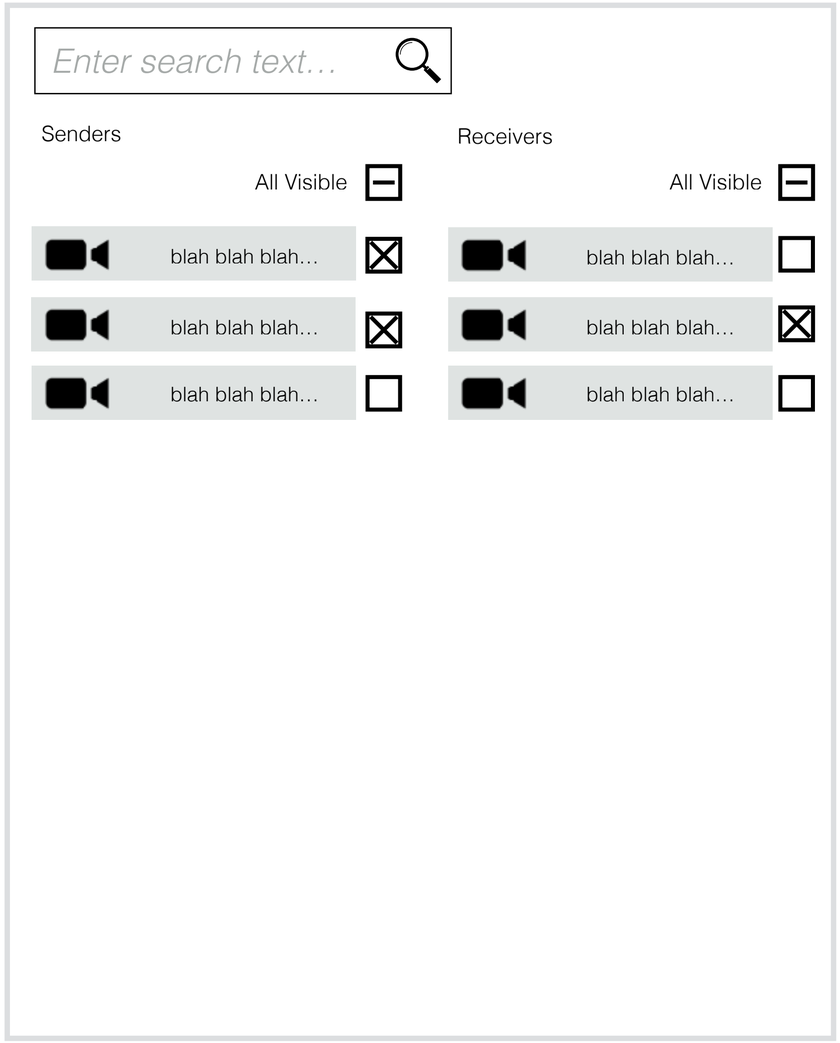
Choose



If either list of routables is longer than the available vertical space, the routables (including the “all visible” meta-routable) and their column headings form a scrollable view, while the search box and buttons remain fixed in position.



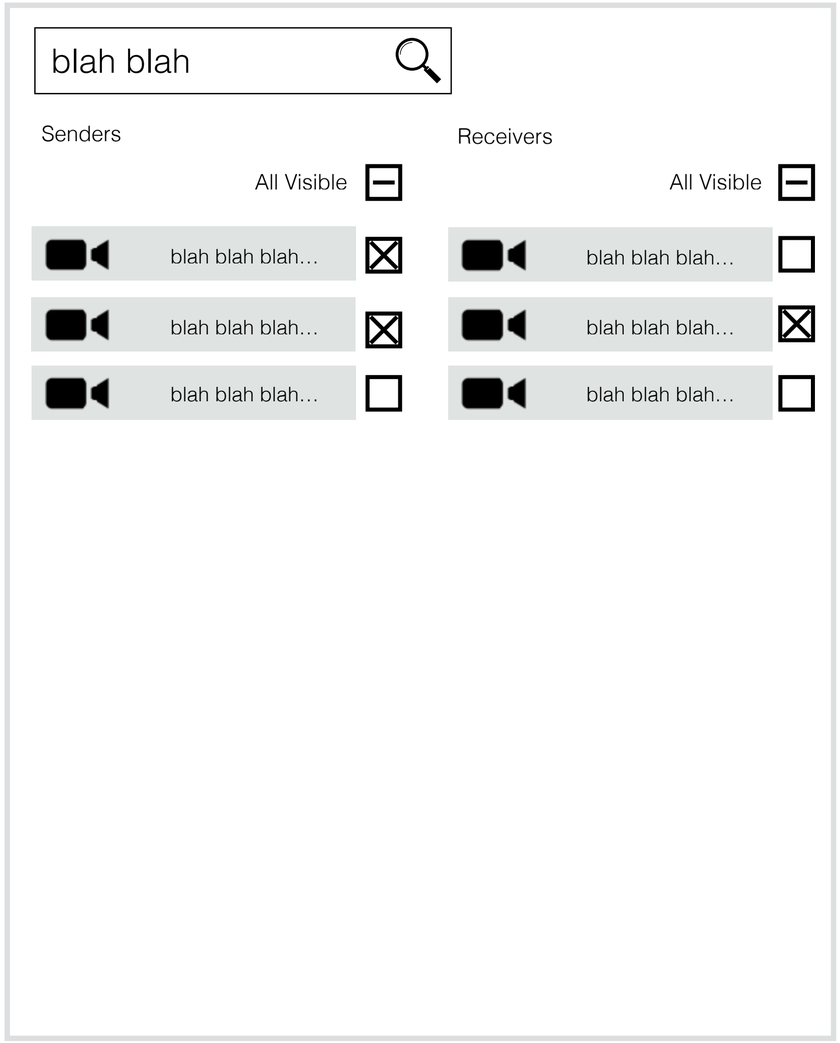
Choose



When the user selects or deselects a routable, that routable immediately appears or disappears in the “Route” view (animate as per routables appearing and disappearing on the network, see section on “notification of changes”). The “all visible” meta-routables update to reflect the checked-ness of the routables in their list: checked if all the routables are checked, unchecked if all are unchecked, and indeterminate if some are checked and some are not.

When the user selects or deselects an “all visible” meta-routable, it and all the routables in its list change their checked state according to the mapping all-visible(checked, unchecked, indeterminate) -> all-in-list(unchecked, checked, checked)..

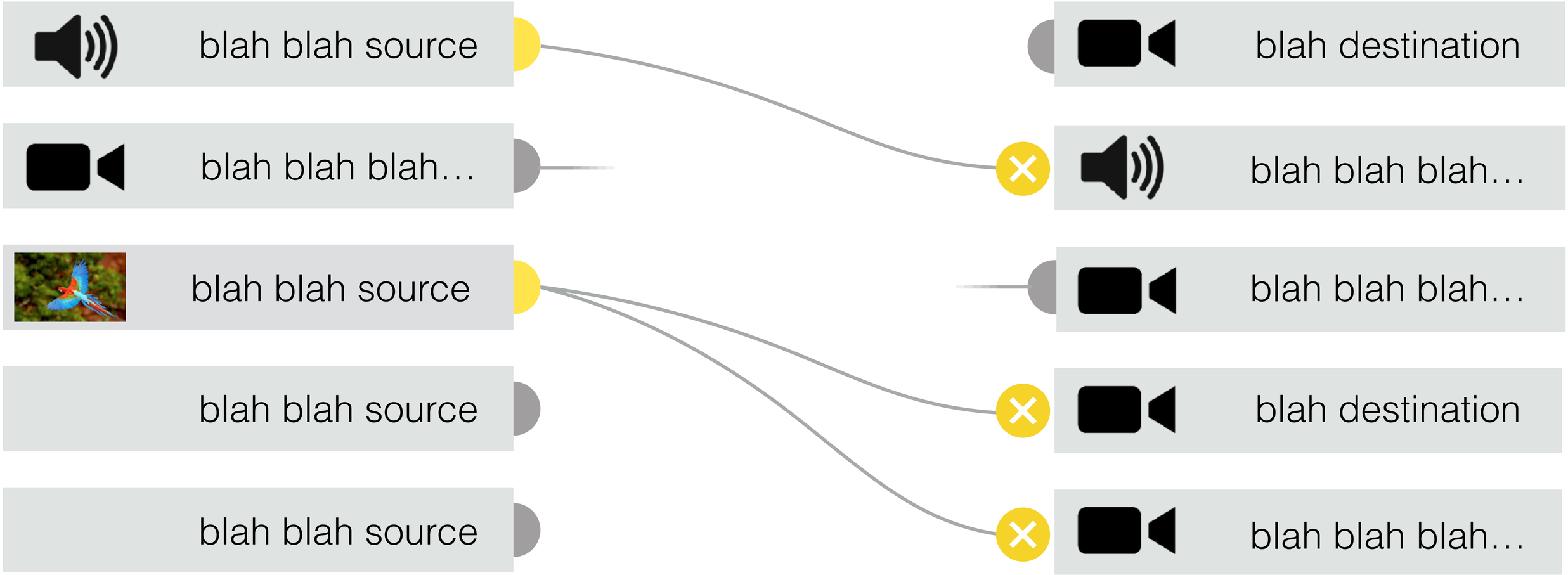
Choose



When the user enters, alters or removes text in the search box, the lists update to show only routables for which a case insensitive substring match of the search text against the label or ID of the routable returns true. DO NOT ANIMATE THIS. :-)

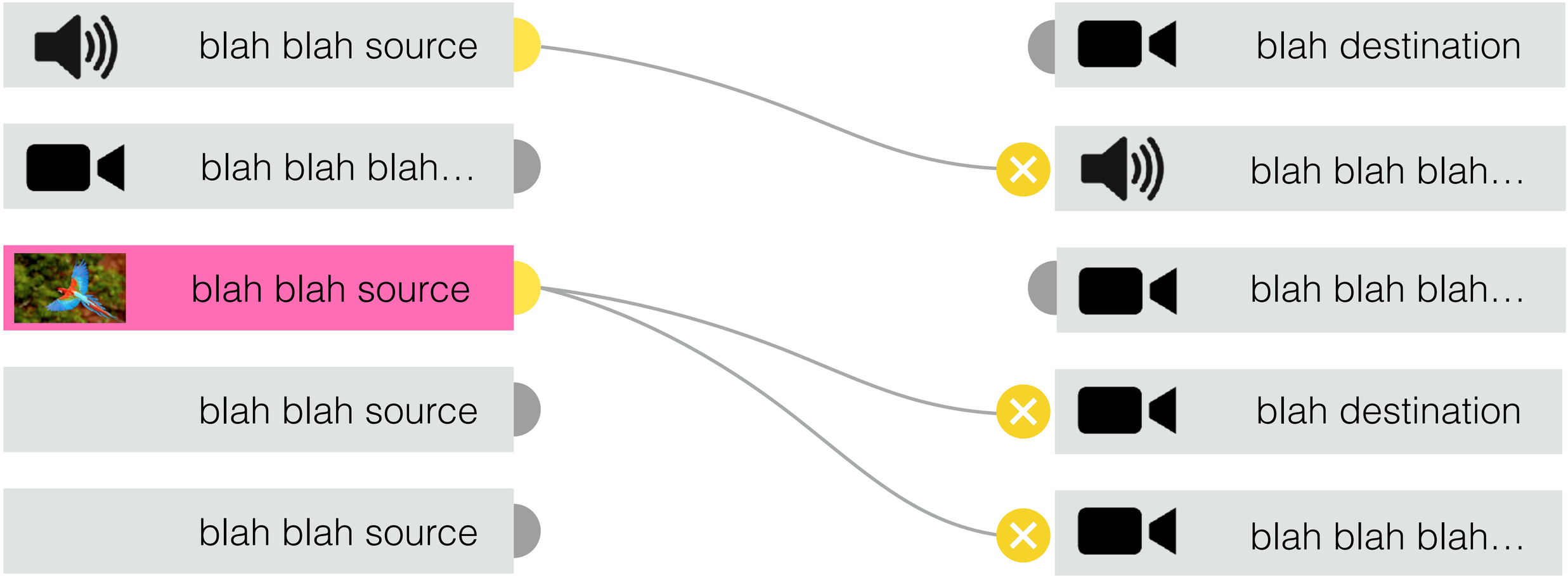
See the “Notification of Changes” section for details of how to update this view when routables appear / disappear on the network.

“Route” View: Displaying Routes

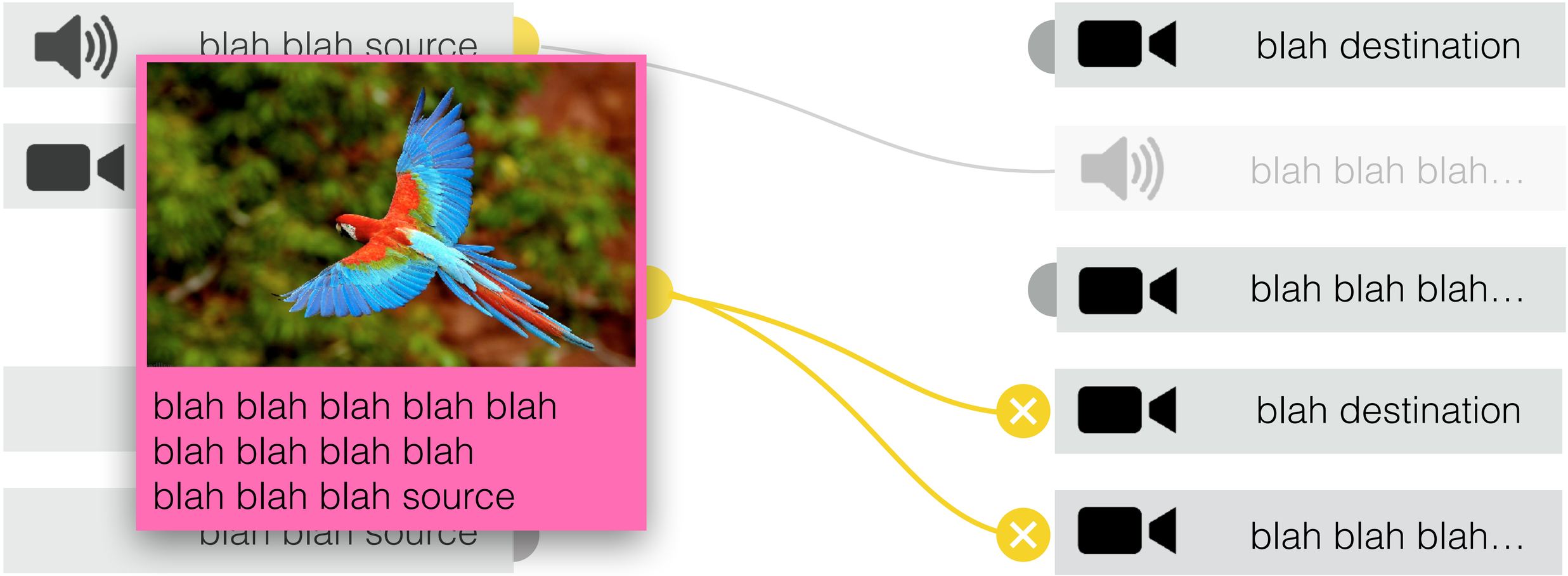


Senders (left) and receivers (right) are represented by grey boxes, with a thumbnail of the current picture (if available) or an icon representing the media type (if not) and the label. Routes between them are shown as curved lines drawn between “nodes” on their inner edges. At the receiver end of a route, the node is shown as a circular “x”, representing a control that the user can use to destroy the route. Sometimes, only one end of a route will be displayable (eg because the user has chosen not to show the other end. In this circumstance, the route is shown as a short horizontal line fading to 0% opacity.

“Route” View:
Creating a Route



The user selects a source by clicking or tapping it.



The source expands, is indented and gains a drop shadow. It now remains stationary when the list scrolls. Its routes are indicated by a thick line the same colour as an active node, rendered on top of all other routes. Other sources fade to 75% opacity. Clicking on another source causes that source to “pop out” and the previous one to regain its former style and place in the list.) Destinations that are incompatible with that source lose their nodes and fade to 25% opacity over a time period of 0.25s. Routes from other senders fade to 25% opacity over a time period of 0.25s

blah blah blah…

blah blah source

blah blah blah blah blah
blah blah blah blah
blah blah blah source

blah blah source

blah blah source

blah blah blah…

blah destination

blah blah blah…

blah destination

blah blah blah…

blah blah blah…

blah destination

The user has scrolled - the selected source stays in the same place.

blah blah blah…

blah blah source

blah blah blah blah blah
blah blah blah blah
blah blah blah source

blah blah source

blah blah source

blah blah blah…

blah destination

blah blah blah…

blah destination

blah blah blah…

blah blah blah…

blah destination

One way to route - the user just clicks the destination.

blah blah blah…

blah blah source

blah blah blah blah blah
blah blah blah blah
blah blah blah source

blah blah source

blah blah source

blah blah blah…

blah destination

blah blah blah…

blah destination

blah blah blah…

blah blah blah…

blah destination

The change has been requested via the API

blah blah blah…

blah blah source

blah blah blah blah blah
blah blah blah blah
blah blah blah source

blah blah source

blah blah source

blah blah blah…

blah destination

blah blah blah…

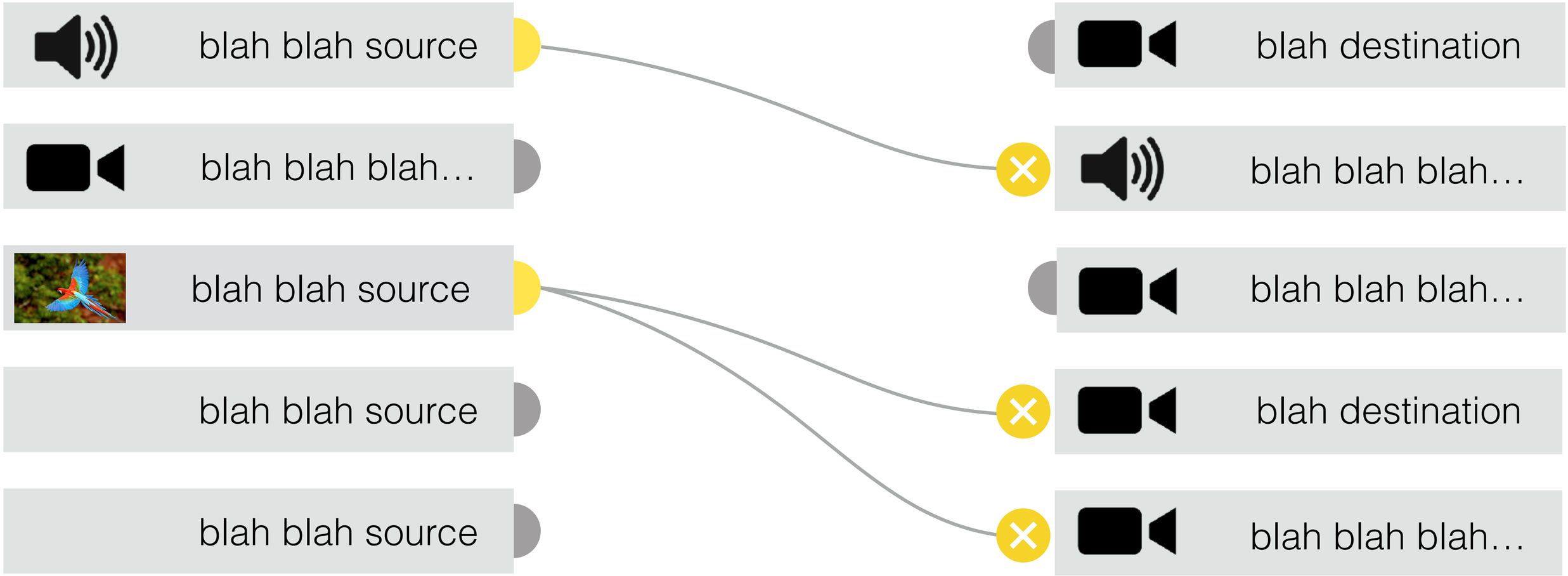
blah destination

blah blah blah…

blah blah blah…

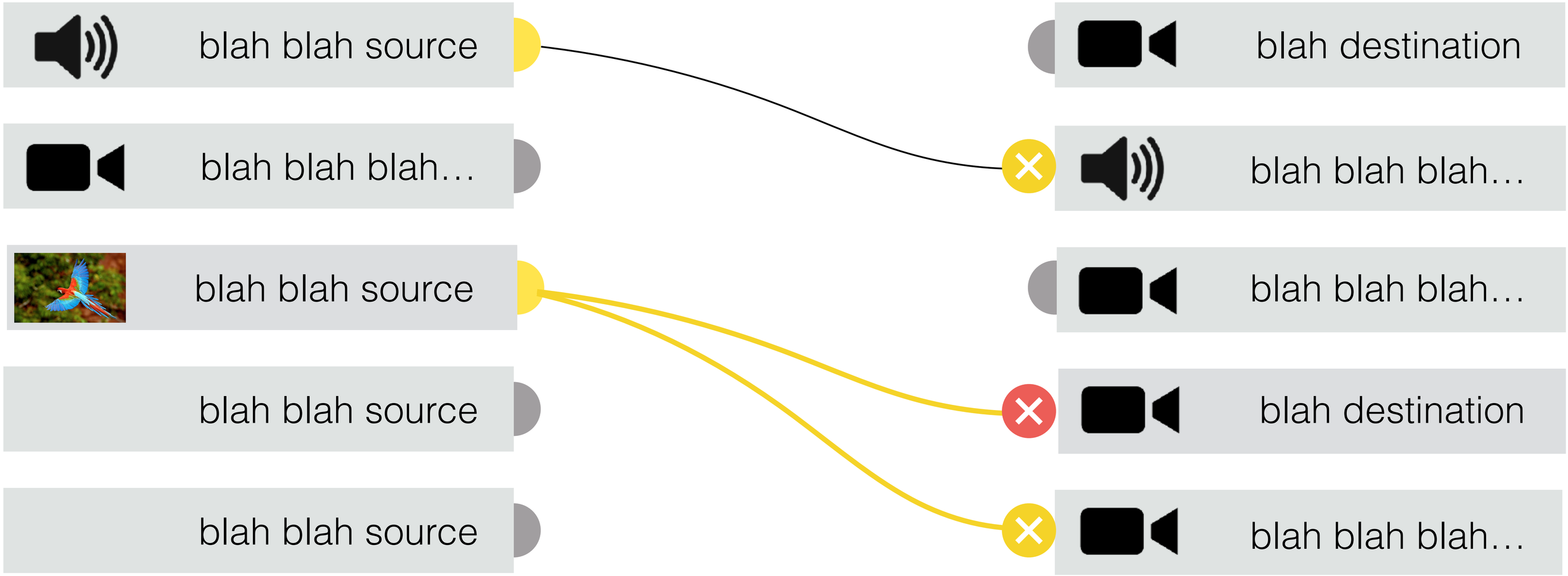
blah destination

The change has been made (feel free to animate to this state with a fade)

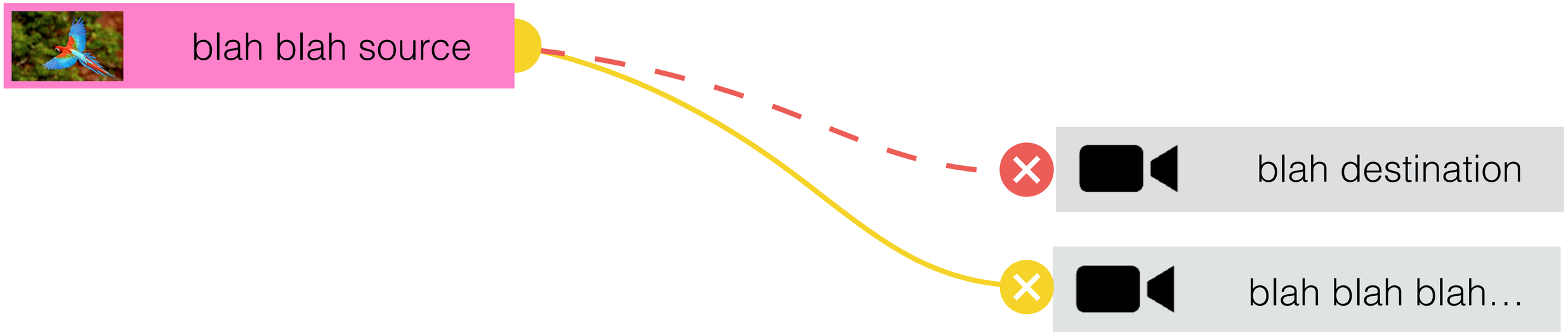


The user clicks or taps anywhere that isn't an active area for another control. The sender returns to its place in the list and loses its highlight. All routes, senders and receivers return to 100% opacity. Nodes of receivers incompatible with the previously selected sender become visible again.

“Route” View: Deleting a Route



The user selects a crossed node for a routed destination by clicking or tapping it.
The node turns red while the mouse / finger is down. The unroute action is triggered
on mouse/finger up within the node's active area. If the user drags their finger/pointer out of the
active area, the node returns to yellow to indicate that an unclick/touch-up event will have



The app sends a ‘destroy’ call to the api. The UI tells the user that it is in the process of destroying the route by turning the line red. The line also becomes dashed.
The source node reverts to the colour it would have been if the selected route had already been destroyed (in this case, yellow, as there is another route from that source.

blah blah source

blah destination

blah blah blah…

Once the destroy operation has been processed the line fades out slowly (~1s), the de
a node back, and it remains red for ~ 2 seconds

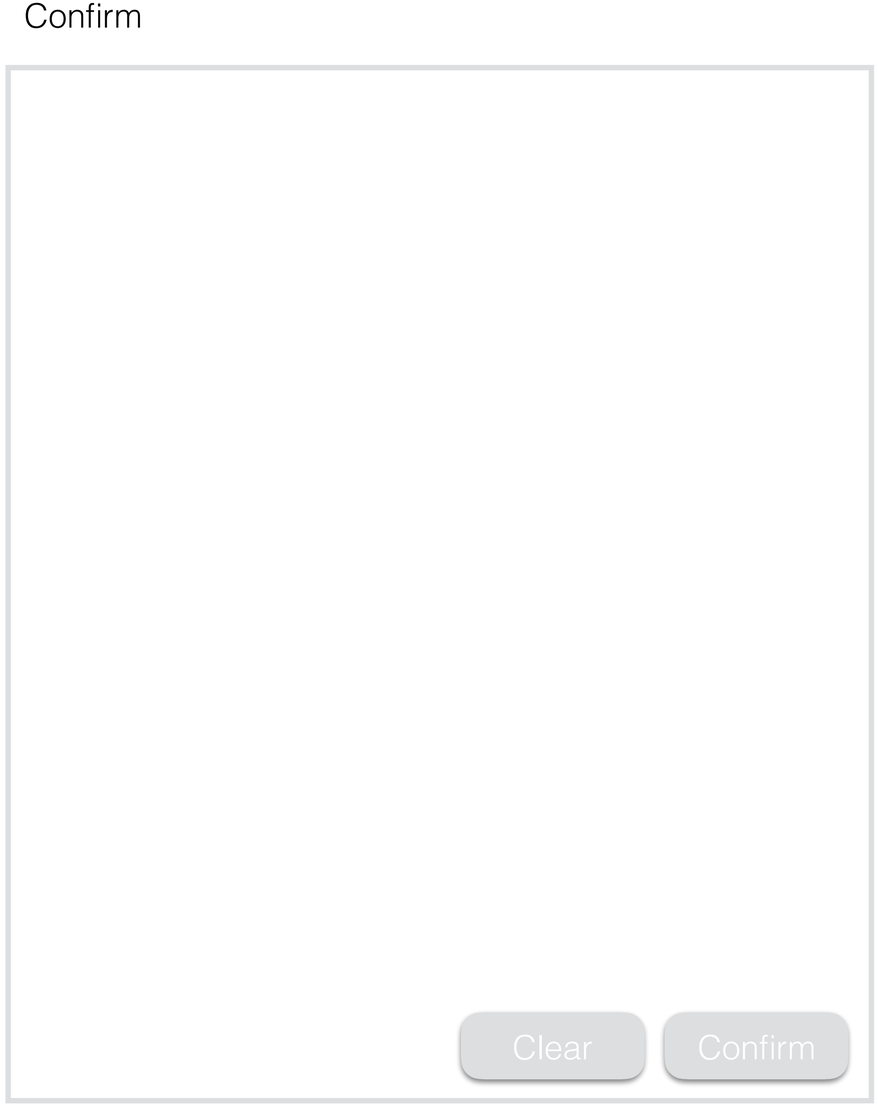
The nodes then lose their red colour, returning to whatever state they would otherwise have had. (The destination will always be grey after an unroute.)

“Confirm” View

When the app is in “confirm changes” mode, the “Confirm” view may be used. (It must be used when in this mode in order to actually route or unroute things.) The purpose of this view is to let the user view and modify a queue of routing changes.

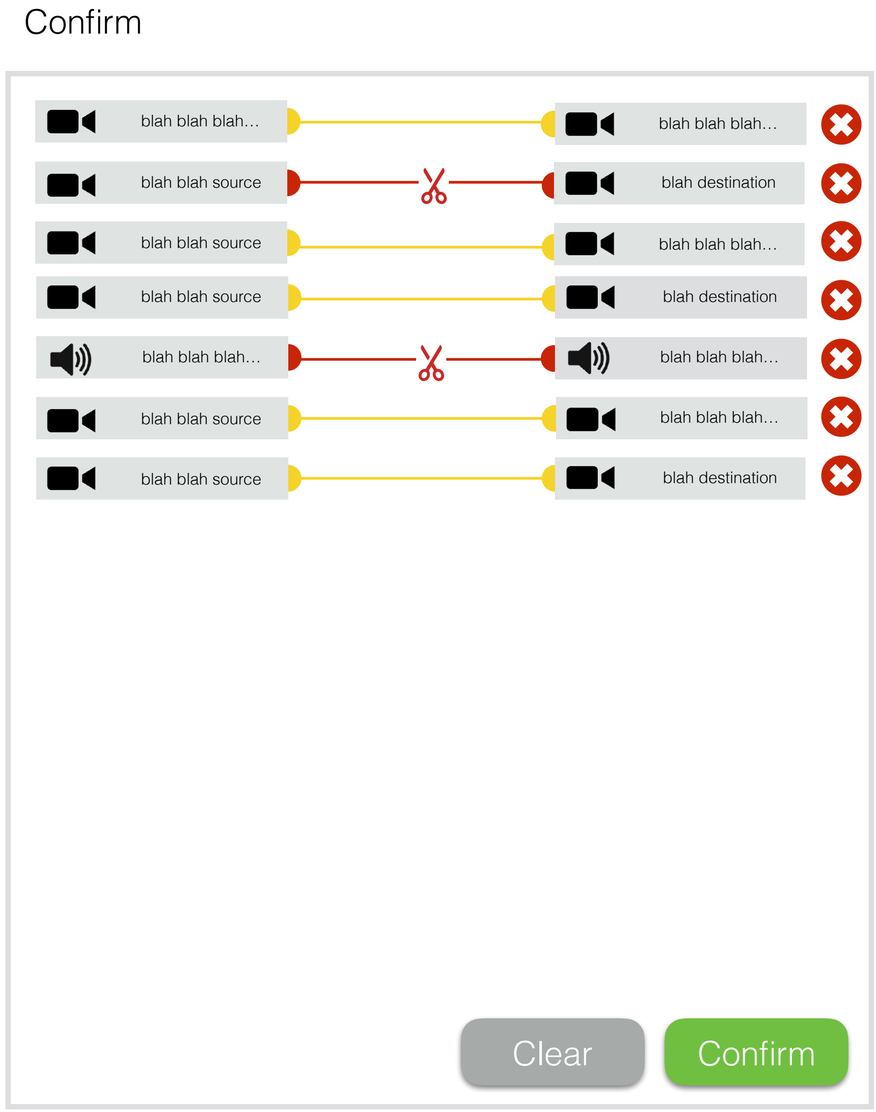
The view consists of a list of proposed changes, and “confirm” and “clear” buttons. If there are too many proposed changes to fit into the view, the user may scroll to see the full list. The buttons remain fixed in position when the list scrolls.

When the user first enables the “confirm changes” view (and on app launch), the list is empty, and the “clear” and “confirm” buttons are disabled. The buttons are enabled when (and only when) there is at least one item in the list.



When “confirm changes” is enabled, every time the user indicates that they wish to create or delete a route, a new row is added to the bottom of the list. Each row contains a representation of the sender and receiver between which a new route is being created or an old one destroyed, a line connecting them that indicates whether the change is creative (yellow) or destructive (red w/ scissors), and a “clear item” node that allows the requested change to be cancelled.

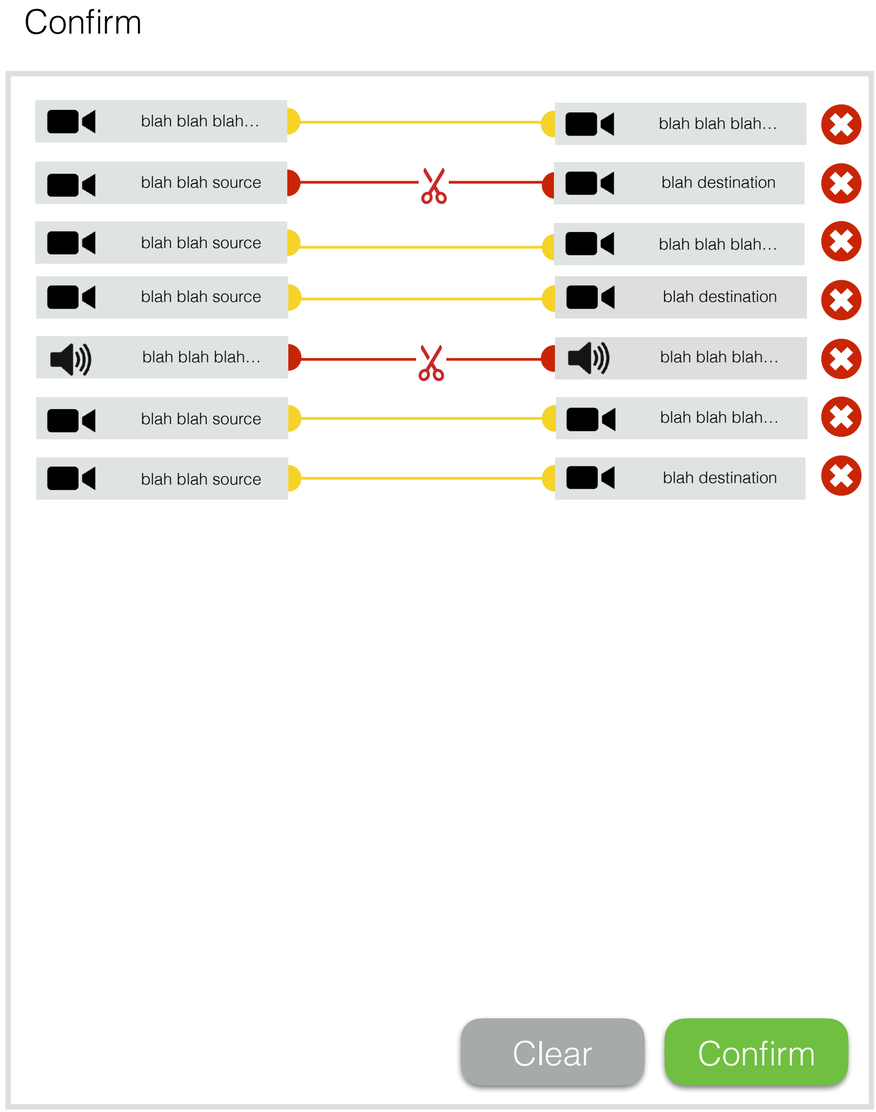
Clicking on the “clear item” node causes the proposed list item to be removed from the list (it fades out (animate 0.25s) and then the list items move to eliminate the gap. Clicking elsewhere in the list item has no effect.



If the user indicates that they want to route or unroute a sender to a given receiver, and then indicates that they want to make a different change to the routing status of that receiver as part of the same set of proposed changes, the new change should be added to the bottom of the list in the usual way. At the same time, the old proposed change should fade out (animate 0.25s) and the list items below its old position (including the new proposed change) should move up to fill the gap.

If the user indicates that they want to make the same change to the routing status of a given receiver more than once in the same set of proposed changes, second and subsequent user interactions should have no effect.

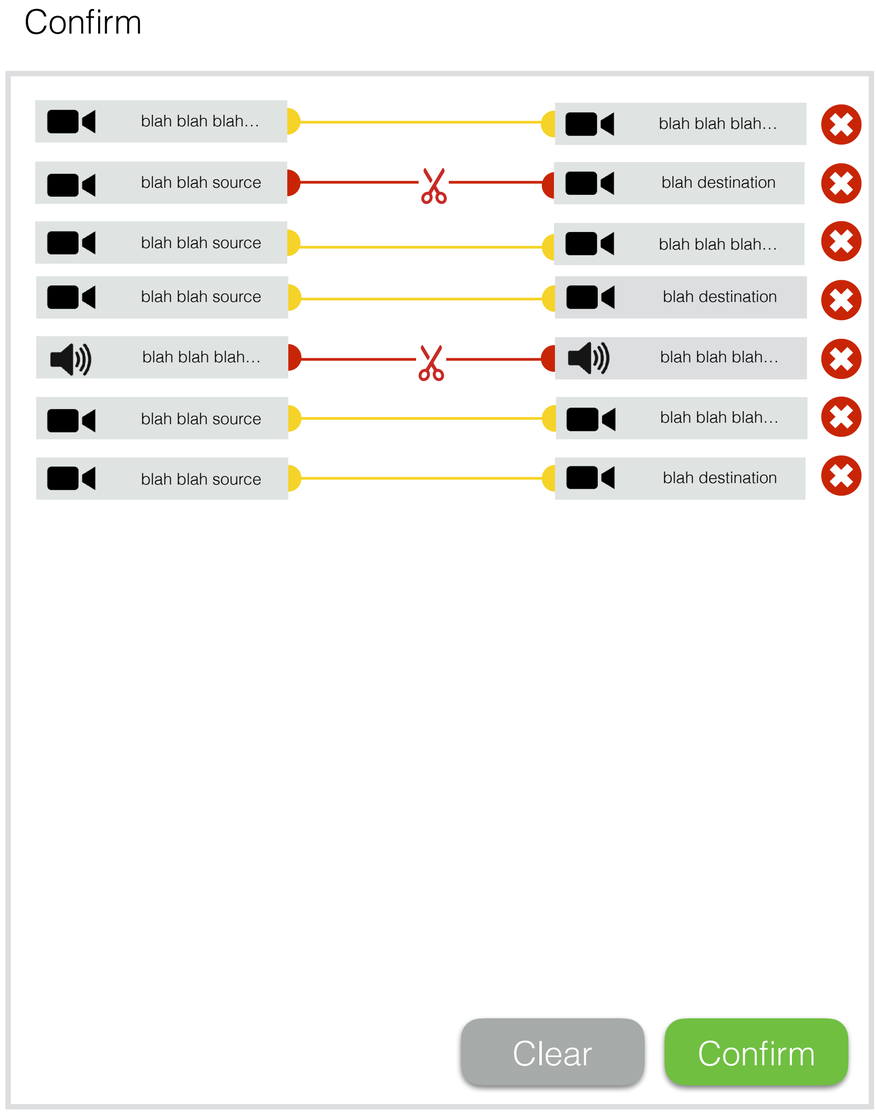
For the avoidance of doubt, a given receiver should never appear more than once in a given set of changes to be confirmed.



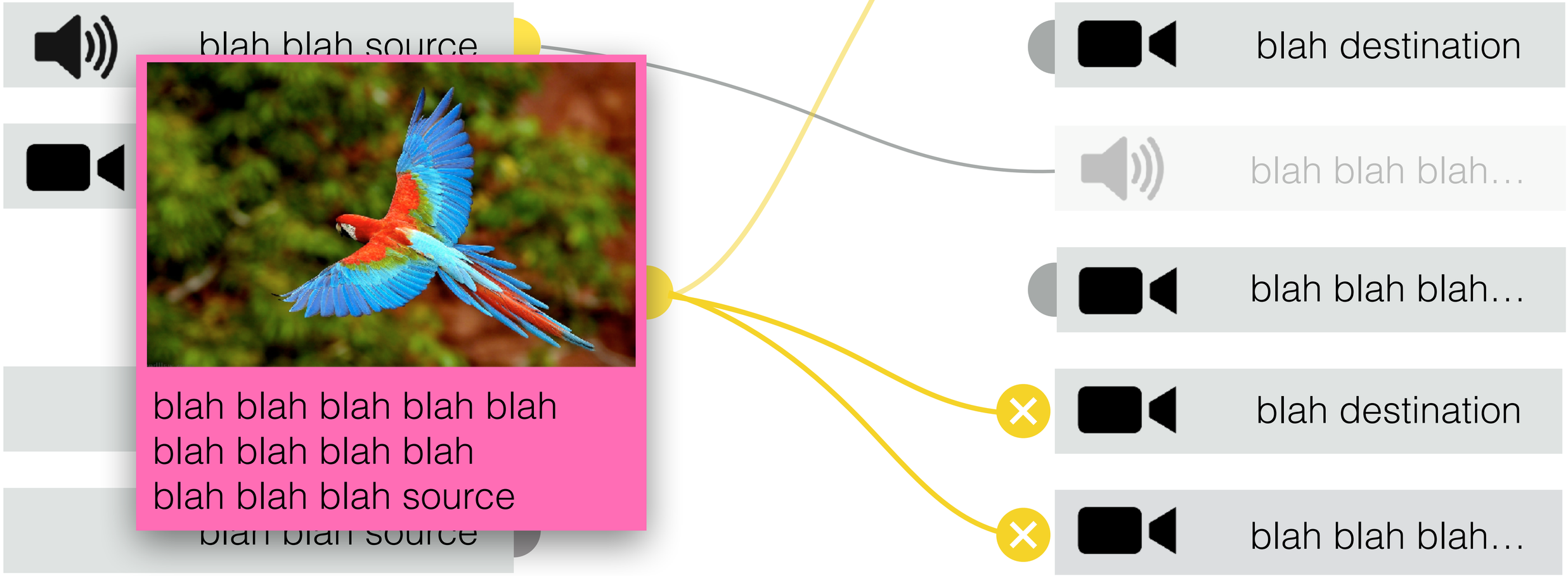
Clicking on the “clear” button causes every item in the list to be removed.

Clicking on the “confirm” button causes every item in the list to be sent to the API as a routing or unrouting request, as appropriate. Once confirmation is received that each request is being processed, the associated list item is removed from the list.

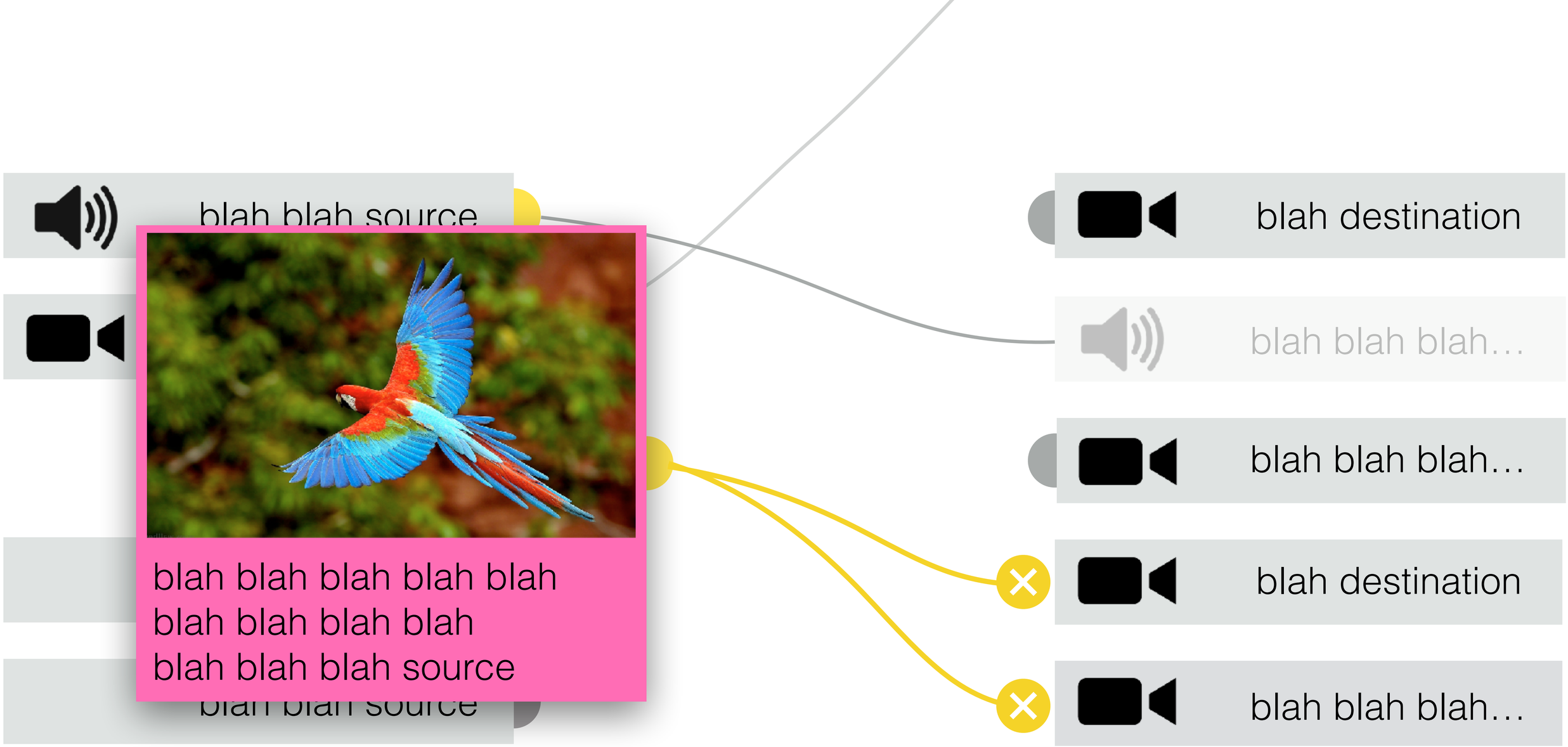
If the API does not accept some or all of the requests, their associated list items remain in the list and the user is notified of the failure.



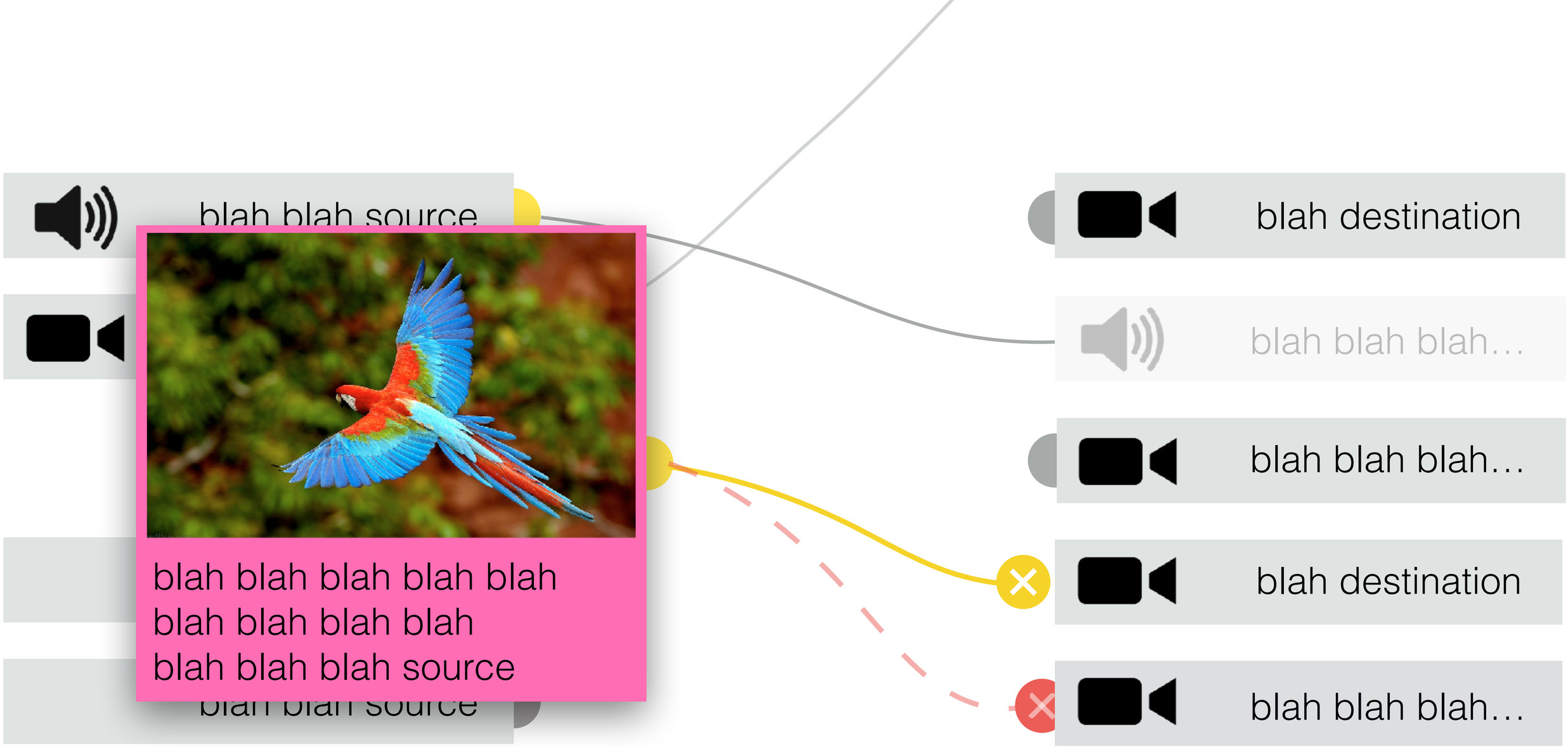
Notifications of changes



A notification is received from the subscription API that a new route is available. The UI indicates this to the user by adding it to the view, animating the addition with a slow fade (~1s) to the appropriate rendering state. In this case, the new route (shown half-way through the fade) is from the selected sender, so the fade is to a thick yellow line. If a notification is received that the route has been destroyed again before the animation has completed, the animation reverses and fades back out at the same speed.

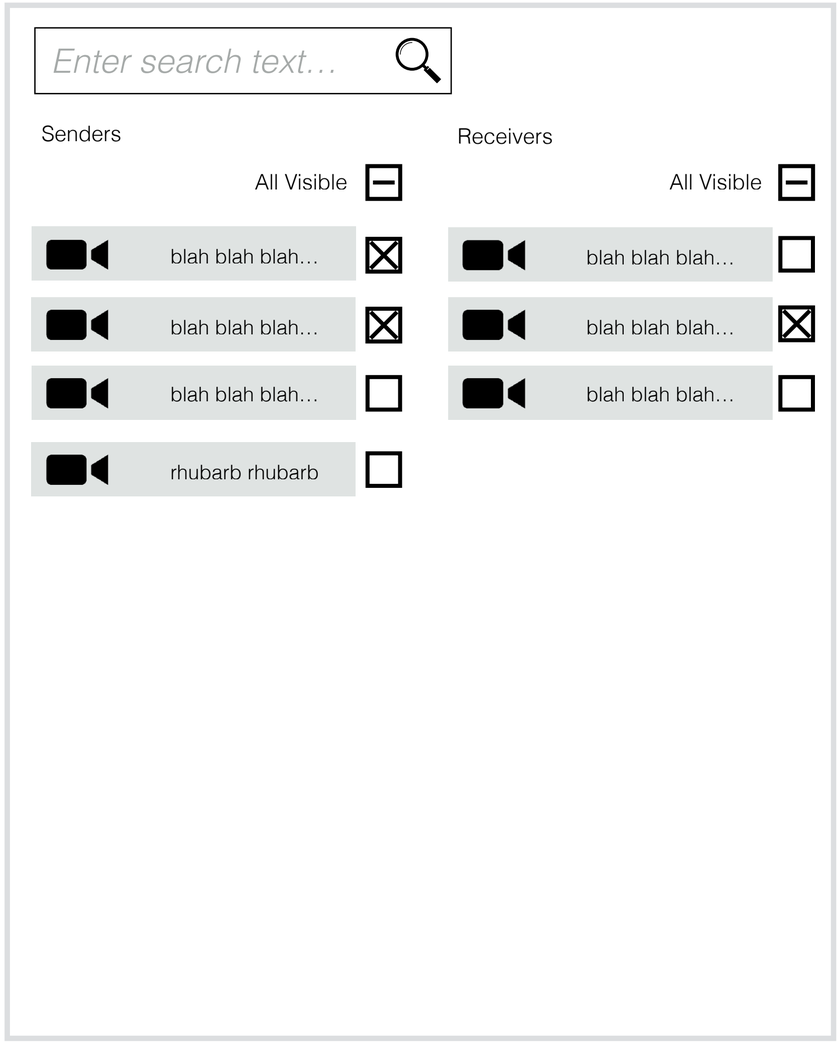


A notification is received from the subscription API that a route has been destroyed. The UI indicates this to the user by removing it from the view, animating the removal with a slow fade (~1s) to the appropriate rendering state. In this case, the destroyed route (shown half-way through the fade) is from an unselected sender, so the fade is from a thin grey line. If a notification is received that the route has been created again before the animation has completed, the animation reverses and fades back in again at the same speed.



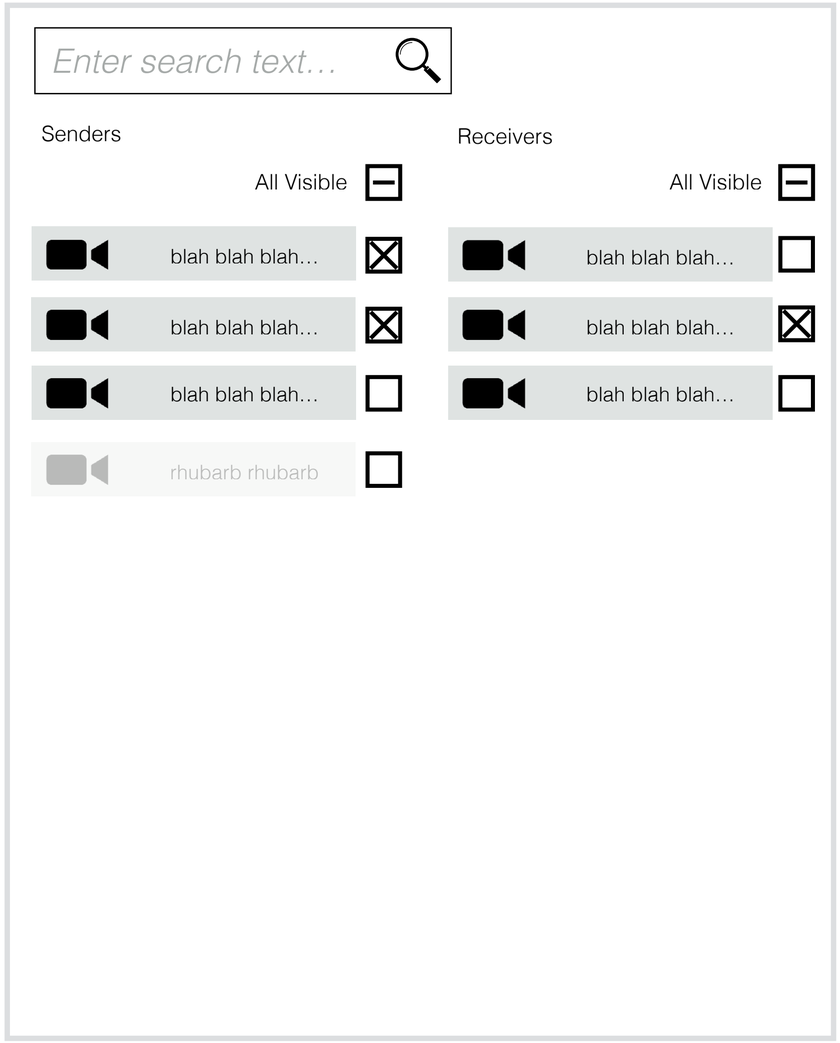
The user has requested that a route be destroyed, and a notification is received from the subscription API that this action has taken place (perhaps in response to the user request; perhaps coincidentally.) This is indicated to the user as if their request had been processed successfully, regardless of the cause of the change. This is indicated by the line fading from red to grey and disappearing completely. The subsequently unattached node reverts to yellow (as it is still available for re-routing).

Choose

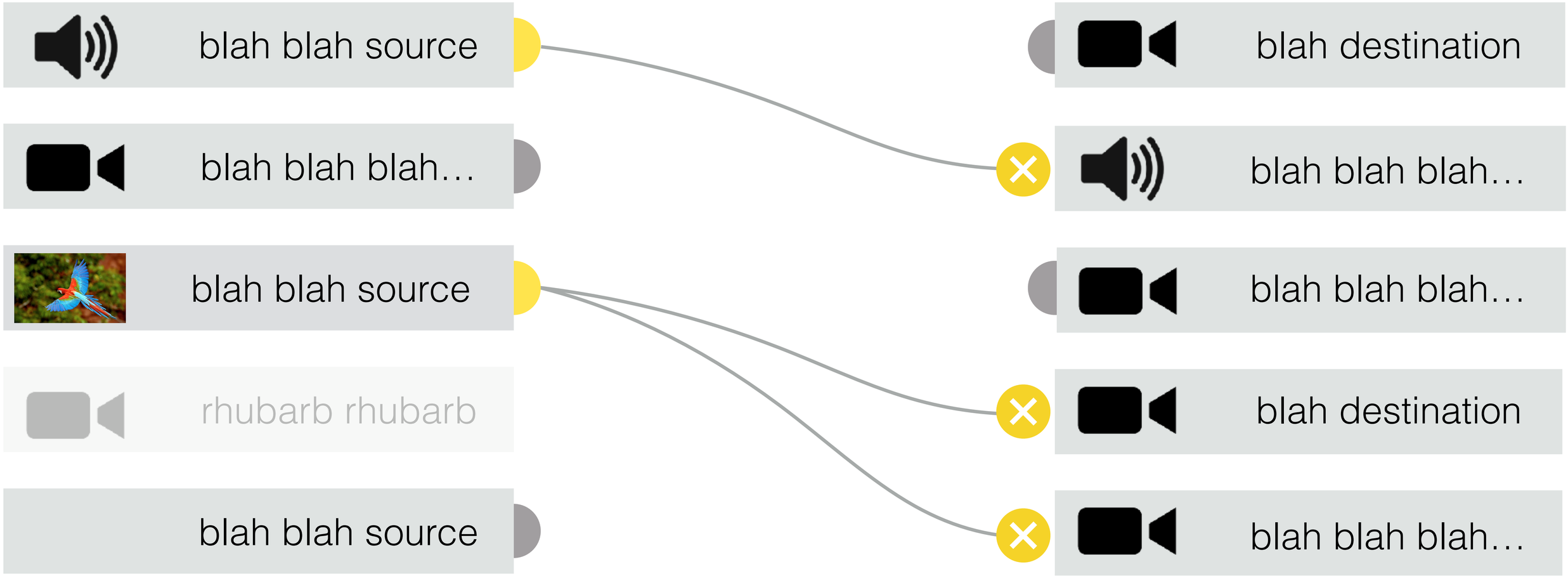


The application receives a notification that a new sender, “rhubarb rhubarb”, is available. If it matches the current search term, it is inserted into the list at the appropriate position. The items below its position in the list translate to their new position (animate 0.25s) and the new sender then fades in (animate 0.5s). The new sender is deselected by default and doesn’t appear in the “Route” view until the user selects it in the “Choose” view. New receivers are treated the same way.

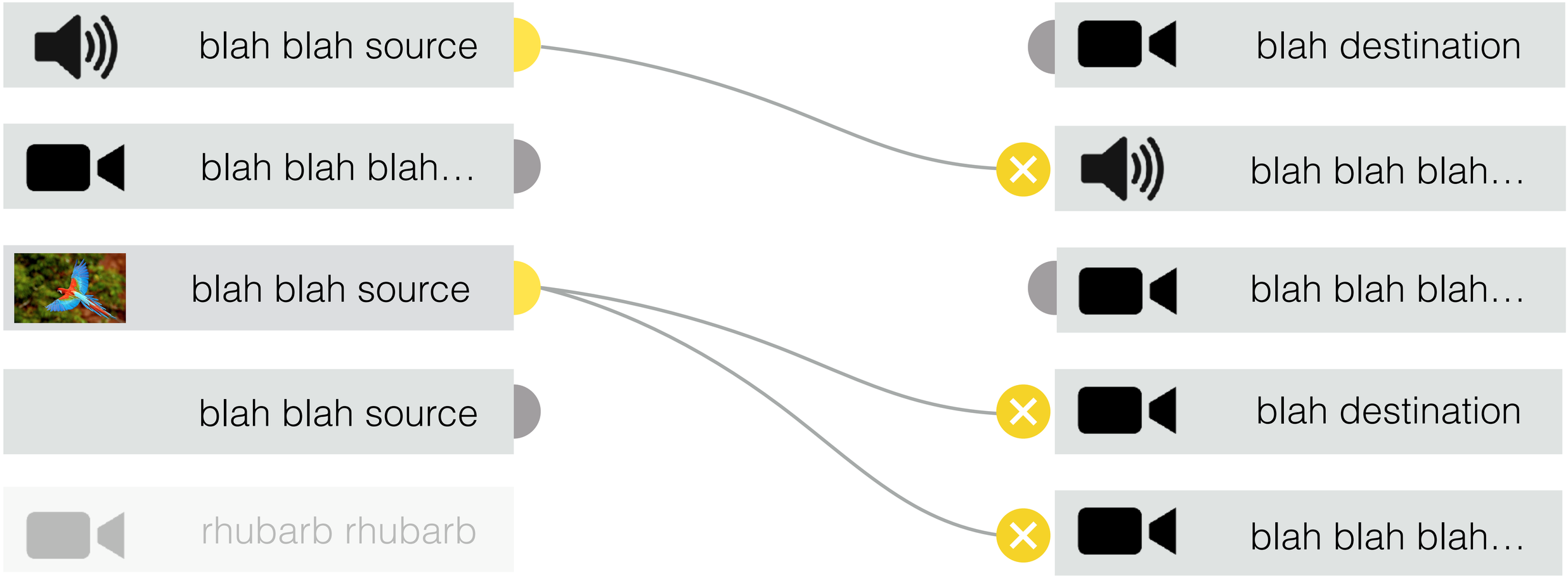
Choose



The application receives a notification that the sender “rhubarb rhubarb” is no longer available. It fades to 25% opacity, but the user can still choose whether or not to show it in the “Route” view. In the event of a page refresh, it is no longer shown. Unavailable receivers are treated the same way.



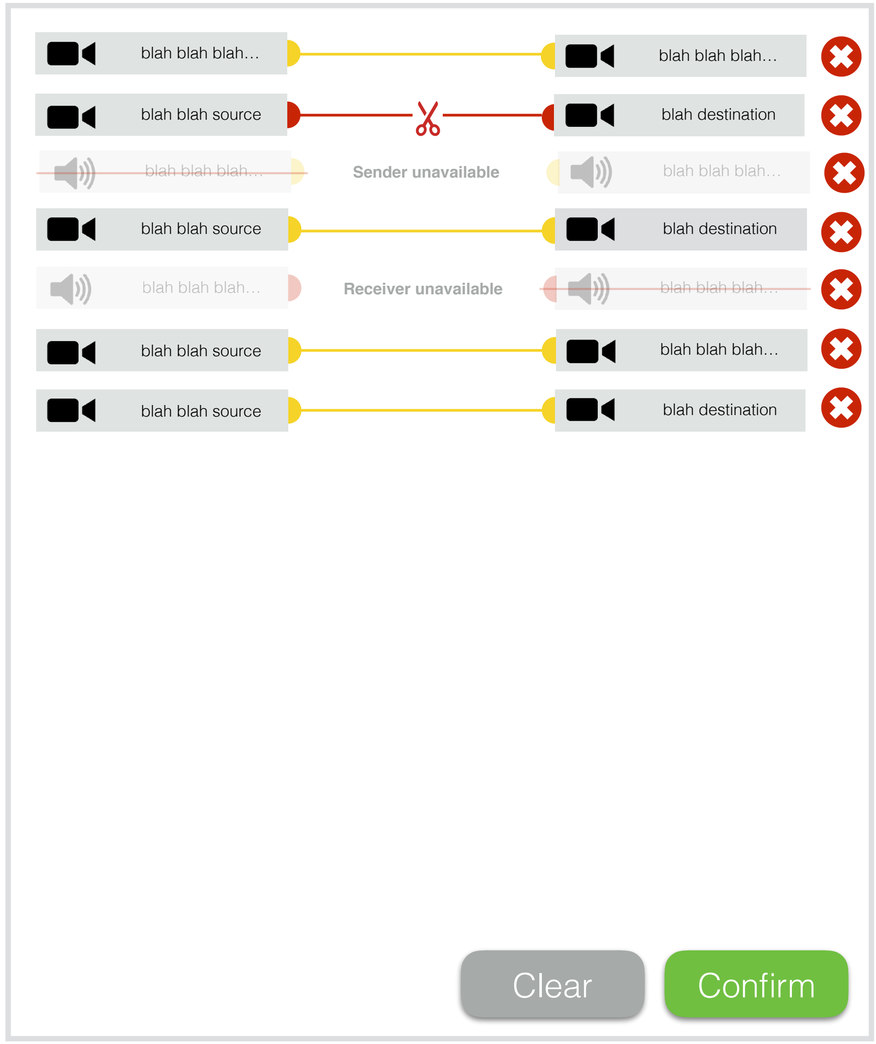
In the “Route” view, if a sender is visible when a notification is received that it is no longer available then i
loses its node, and fades to 25% opacity; its routes fade out completely (0.25s). It can no longer be
selected. In the event of a page refresh, it is no longer shown. Unavailable receivers are treated the same



After one minute, the unavailable sender moves to the bottom of the list. The senders beneath its former position in the list move up to fill the gap. (Animate both, 0.5s.) It can not be selected. In the event of a page refresh, it is no longer shown. If more than one unavailable senders are in this state, they are sorted according to the order of the rest of the list (ie “unavailable for >1 minute”) becomes the first sort critereon. If the sender becomes available again, it moves back into its correct position in the list of available senders (animate, 0.5s) and regains its original styling. Receivers are treated the same way.

In the “confirm” view, if either sender or receiver (or both endpoints) become unavailable then their representation is faded to 25% opacity. The line between the nodes disappears and is replaced by text communicating the problem ‘Sender unavailable’.
A new red line appears which is a strikethrough on the unavailable node.
The ‘clear item’ node are still visible and can be used in the same way as before.

Confirm



Error handling

- Server is down when request is made.
- Server returns invalid request error immediately.
- Unable to parse JSON in server response.
- Server returns a different error

- For all of the above, dump error details to the console, and fade the line relating to the request back to whatever it was before the request was signalled. Show a modal dialog to the user giving as much information about the error as possible. If the user had 20 requests queued up, don't show 20 dialog boxes!

- Websocket disconnection immediately after request.

- Try reconnecting for up to one minute.

- We aren't seeing an update following a request.

  - One minute after each request, if the web socket has never given us an update for the request, fade the line relating to the request from the "in-progress" state to whatever it was before. Remove the associated list item from the "confirm" list (if applicable).

- Websocket disconnection at some other point in time.

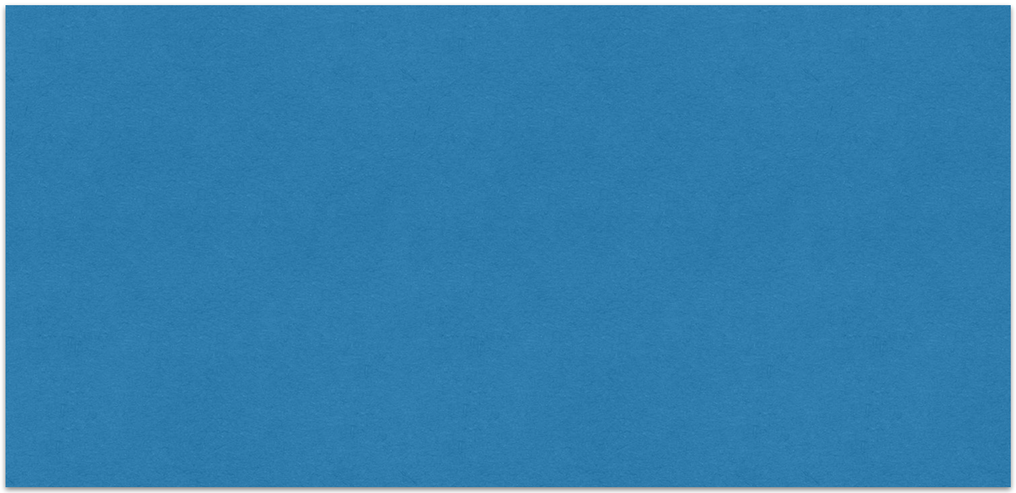
- Try reconnecting for up to one minute.

Web Router

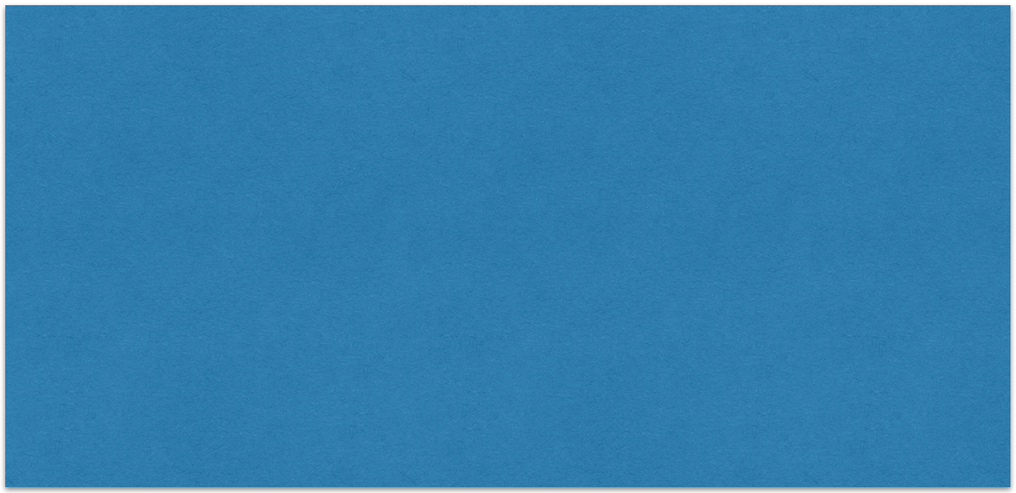
! Disconnected… trying to reconnect.

Choose | Route | Confirm

Choose



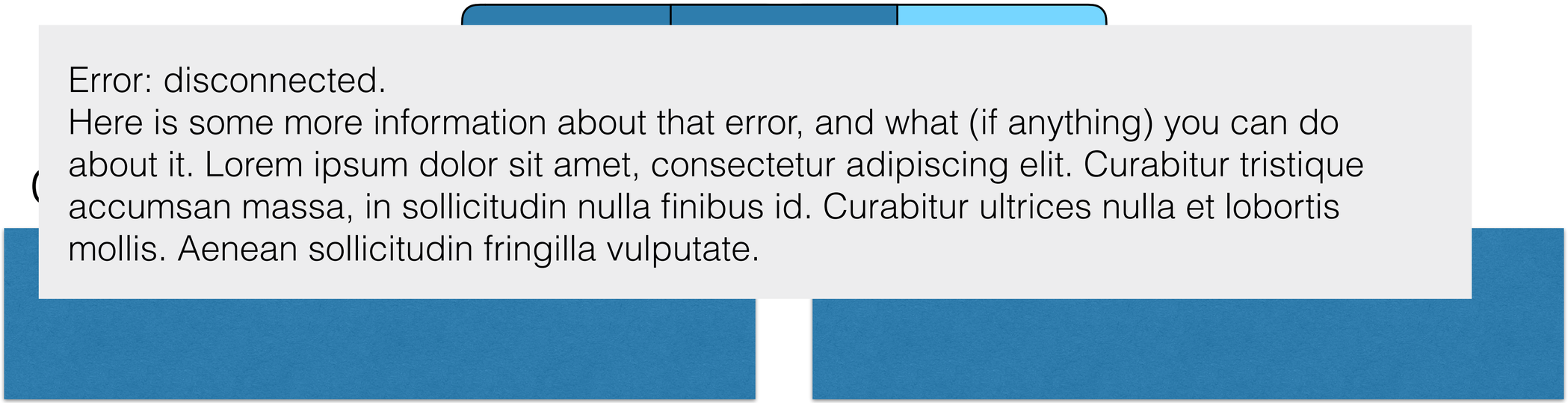
Route



lots of kinds of error messages appear in error notification (only one at a time)

Web Router

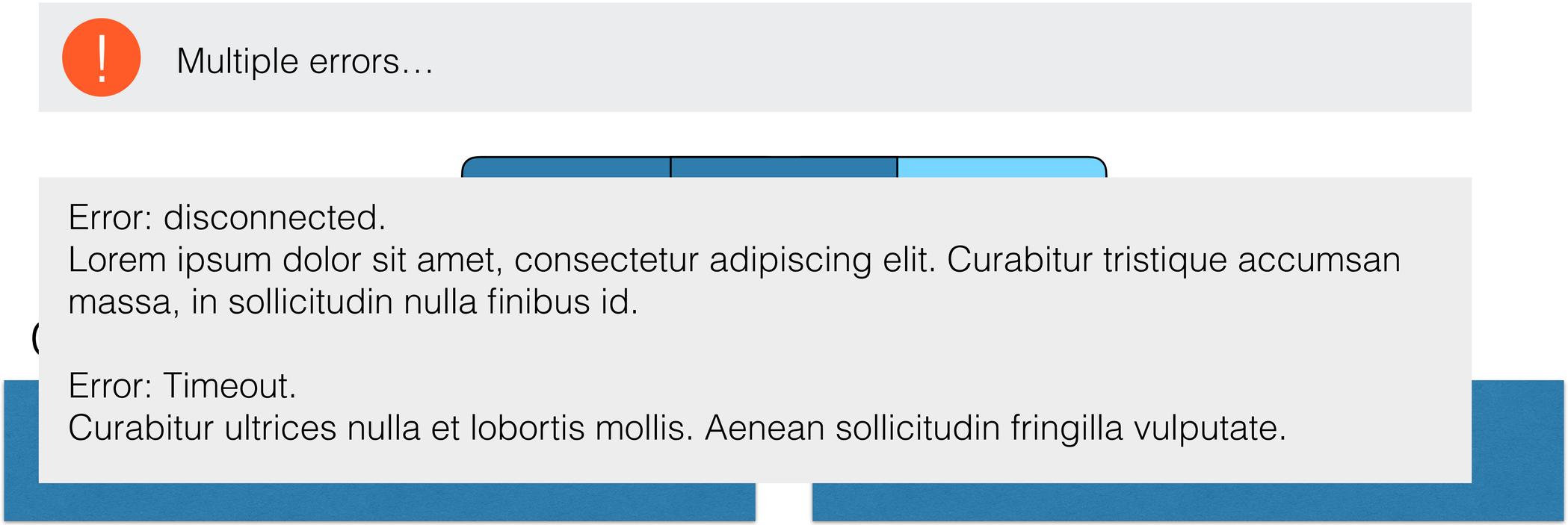
Disconnected… trying to reconnect.



Error: disconnected.
Here is some more information about that error, and what (if anything) you can do about it. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Curabitur tristique accumsan massa, in sollicitudin nulla finibus id. Curabitur ultrices nulla et lobortis mollis. Aenean sollicitudin fringilla vulputate.

interact with this notification to open a dialog with more information about the error.

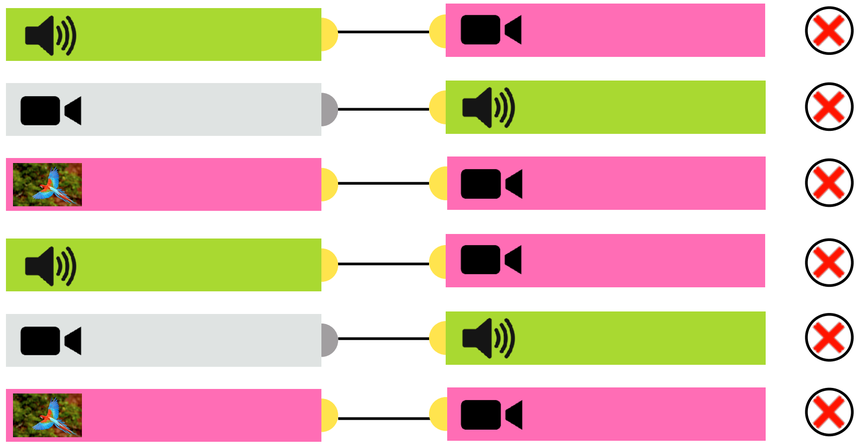
Web Router



If a batch operation results in more than one error, these are combined into one notification (where possible) and communicated to the user as one dialog.
Dismissing the notification, on errors where there is no UI resolution, then we should apply a timeout (30s)

Ignore everything after this.

CONFIRM



blah blah blah…

blah blah source

blah blah blah blah blah
blah blah blah blah
blah blah blah source

blah blah source

blah blah source

blah blah blah…

blah destination

blah blah blah…

blah destination

blah blah blah…

blah blah blah…

blah destination

Another way to route is by dragging a source node to a destination.
The line is shown dashed to indicate that a user request is being made,

blah blah blah…

blah blah source

blah blah blah blah blah
blah blah blah blah
blah blah blah source

blah blah source

blah blah source

blah blah blah…

blah destination

blah destination

blah blah blah…

blah blah blah…

blah destination

A user can visually check a destination by dragging a source node over to one, and hovering above it.

blah blah blah…

blah blah source

blah blah blah blah blah blah blah blah blah blah blah blah source

blah blah source

blah blah source

blah blah blah…

blah destination

blah blah blah…

blah destination

blah blah blah…

blah blah blah…

blah destination

On release the source & destination are connected. A line is drawn between them - dashed to indicate that the connection is in progress.

blah blah blah…

blah blah source

blah blah blah blah blah
blah blah blah blah
blah blah blah source

blah blah source

blah blah source

blah blah blah…

blah destination

blah blah blah…

blah blah blah…

blah blah blah…

blah blah blah…

blah destination

The destination moves back to its position in the list straight away.

blah blah blah…

blah blah source

blah blah blah blah blah
blah blah blah blah
blah blah blah source

blah blah source

blah blah source

blah blah blah…

blah destination

blah blah blah…

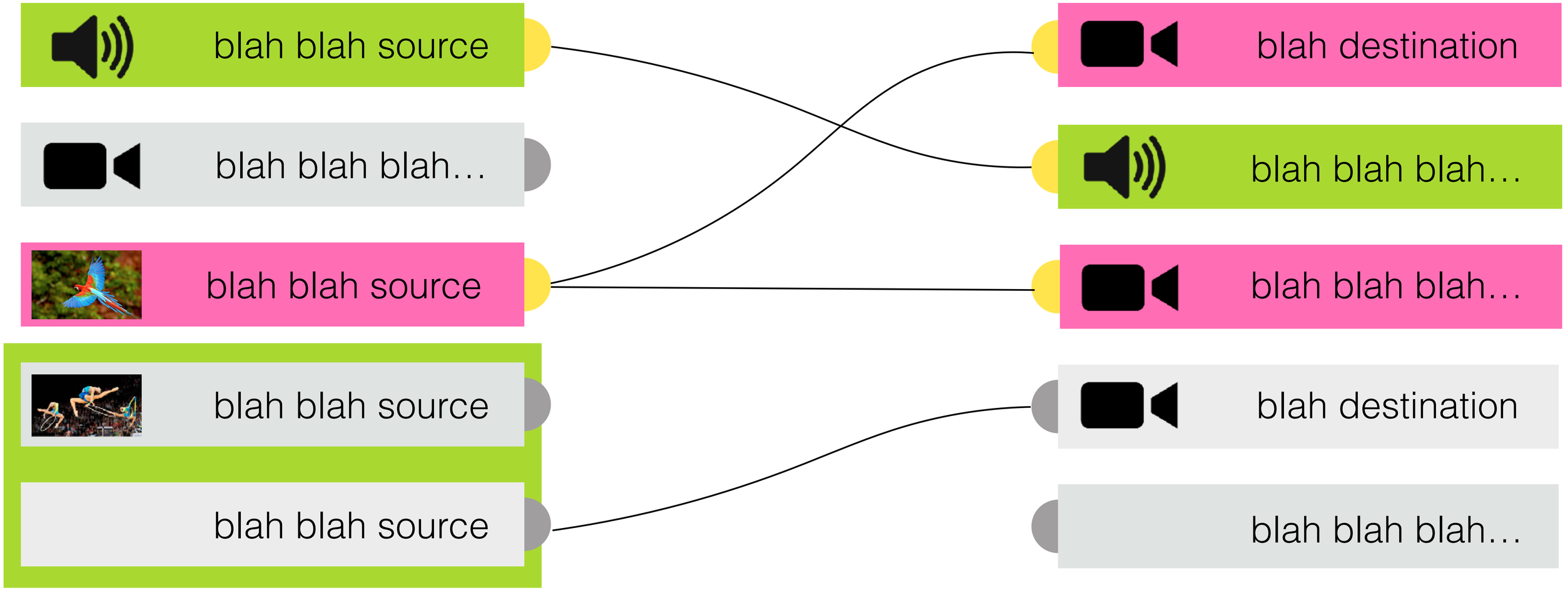
blah blah blah…

blah blah blah…

blah blah blah…

blah destination

The route has been created.

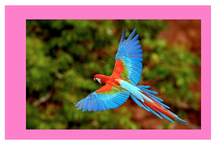

blah blah source
blah blah blah…
blah blah source
blah blah source
blah blah source
blah destination
blah blah blah…
blah blah blah…
blah destination
blah blah blah…

blah blah source

blah blah source

blah blah blah blah blah
blah blah blah blah
blah blah blah source

blah blah blah blah blah
blah blah blah blah
blah blah blah source


blah blah source


blah blah source


blah blah source


blah blah source


blah blah source

blah destination

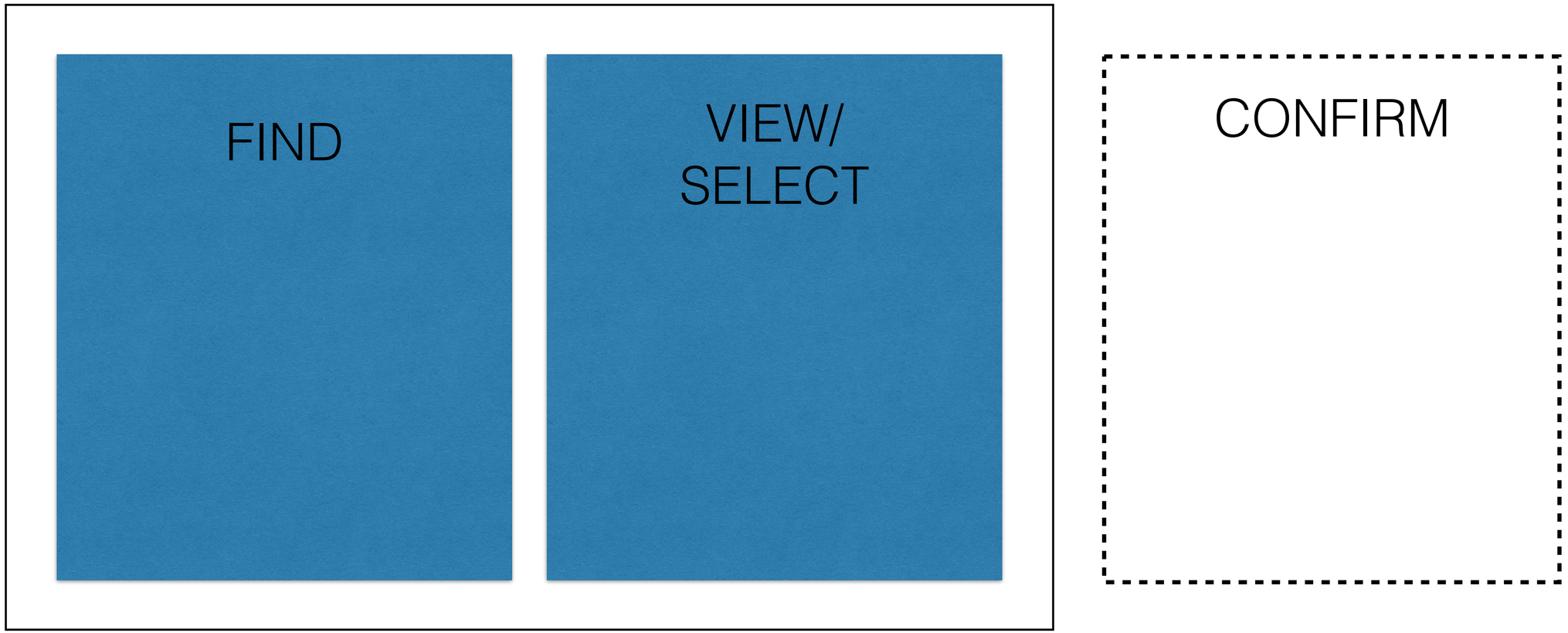
blah destination

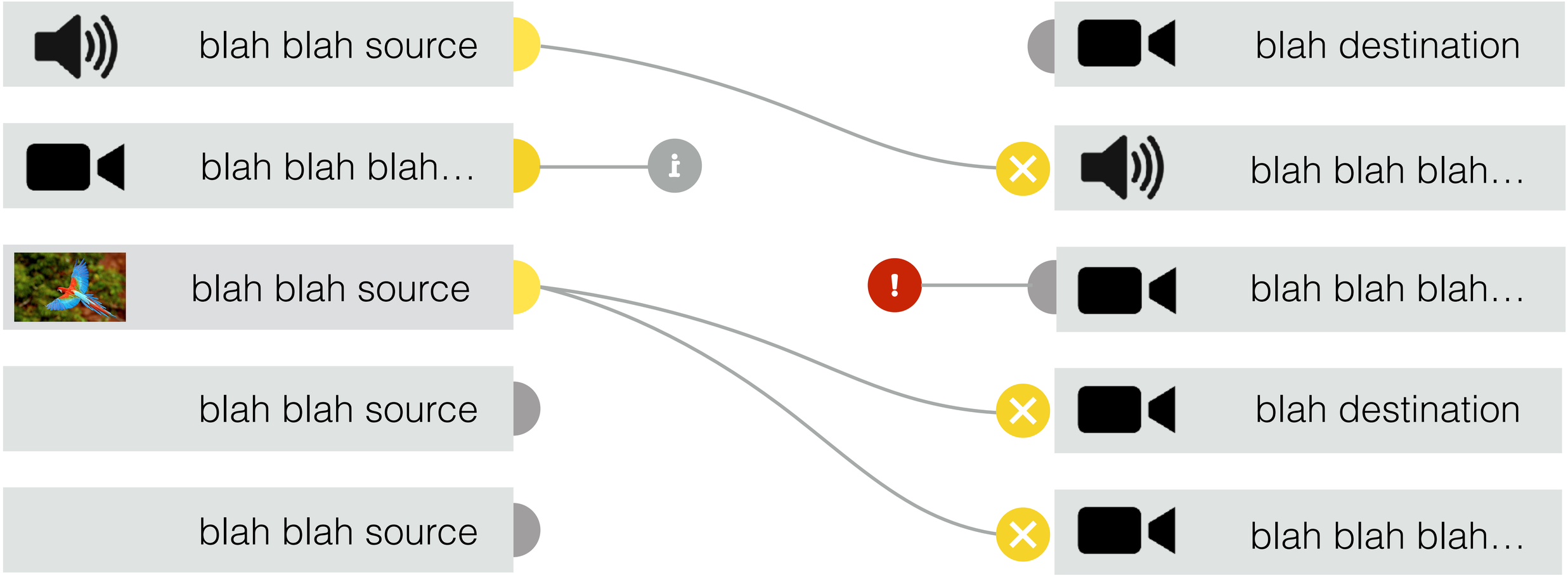
blah destination

blah destination

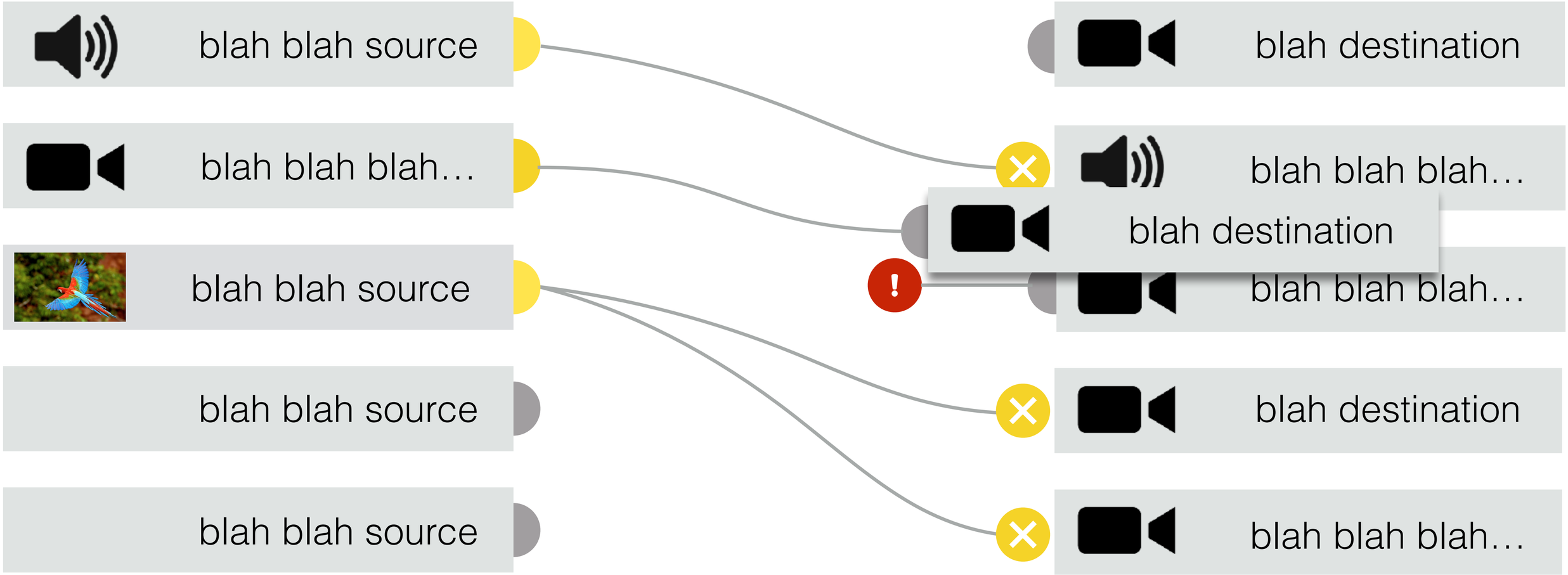
blah destination

blah destination


FIND
VIEW/
SELECT
CONFIRM



In certain circumstances, a route may exist that cannot be fully displayed. If one of the routables for the route is hidden due to a user deselecting it in the “Choose” view, the route can be displayed as being between the visible node and an “information” node rendered towards the middle of the route display area. If a routed routable is hidden due to an error condition, the route can be displayed as being between the visible node and an “error” node rendered similarly.



The user can click/touch the info node to temporarily display the other end of the route, rendered “above” the view like a “selected sender”.

The user can click/touch the error node to bring up a dialog box like the notification dialog that displays details of the error.