Reachability & Failure Detection

draft-ietf-shim6-failure-detection-02.pdf

Jari Arkko
Iljitsch van Beijnum

IETF-64

Outline

- Status & history
- Protocol walkthrough
- Issues

Status and History

- Draft 02 is largely in good shape
- Except for the actual protocol behaviour…

History of the Protocol

- Original version in -00 from prior IETFs
  - Details largely missing
- Iljitsch's draft made us understand reachability part better
  - FBD chosen
- Version -01 for the interim attempted to go provide the detailed behaviour
  - Had some trouble writing pseudocode or state machine
  - Critique from the Interim meeting
- Version -02 based on interim discussions
  - Reachability and exploration in the same messages
  - Had some trouble describing a state machine

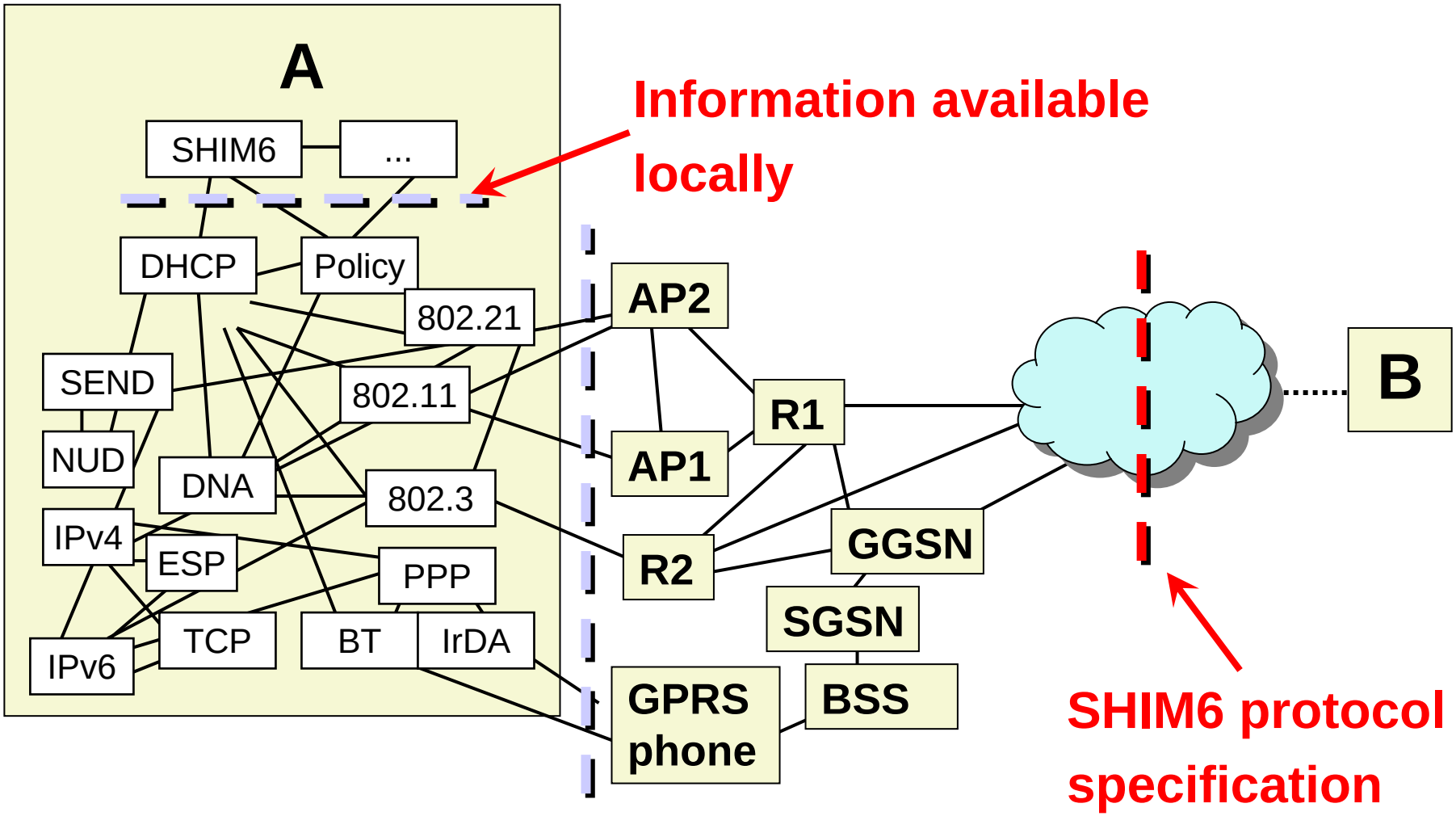
History of the Protocol, Continued

- Erik's simplified description
  - Avoids the state machine
  - Had some trouble writing down all the details
  - Protocol has still holes
- IETF-64
  - Using Erik's model as a basis
  - Attempting to write down the details (not complete yet)

Note: the protocol that we are so hard trying to describe has one (1) message which contains one (1) bit of information relevant for behaviour

Protocol Design & Walkthrough

Scope of the SHIM6 work



Design Decisions

- Multi6 does not go to the area of the configuration modules or protocols -- we shall not reinvent DHCP, and we shall believe what ND tells us
- Own addresses learned locally, peer addresses are communicated
- Multi6 only works as a fail-over, Erik's model:
  - Separate hosts don't share locators to same peer
  - A pair of communicating hosts can have multiple contexts (for separate ULID pairs) with independent locator choices
- FBD is chosen for simplicity
- Sender chooses outgoing address pair (independently from the other direction)

Other Design Goals

- Efficient
  - No packets sent if payload traffic is idle
  - No packets sent if bidirectional payload traffic
  - Packets sent **only** if (a) there's a failure or (b) there's unidirectional traffic
- Handles unidirectional failures
- Construct the protocol so that it provides return routability verification at the same time
- Provide a separable component that might be possible to use in other contexts

Definitions and Background

- Available addresses
- Locally operational address pairs
- Operational address pairs
- Unidirectionally operational address pair
- Current path

Reachability vs. Exploration

- Verifying reachability of the current locator pair(s)
- Exploring for an alternative locator pair when failure is suspected

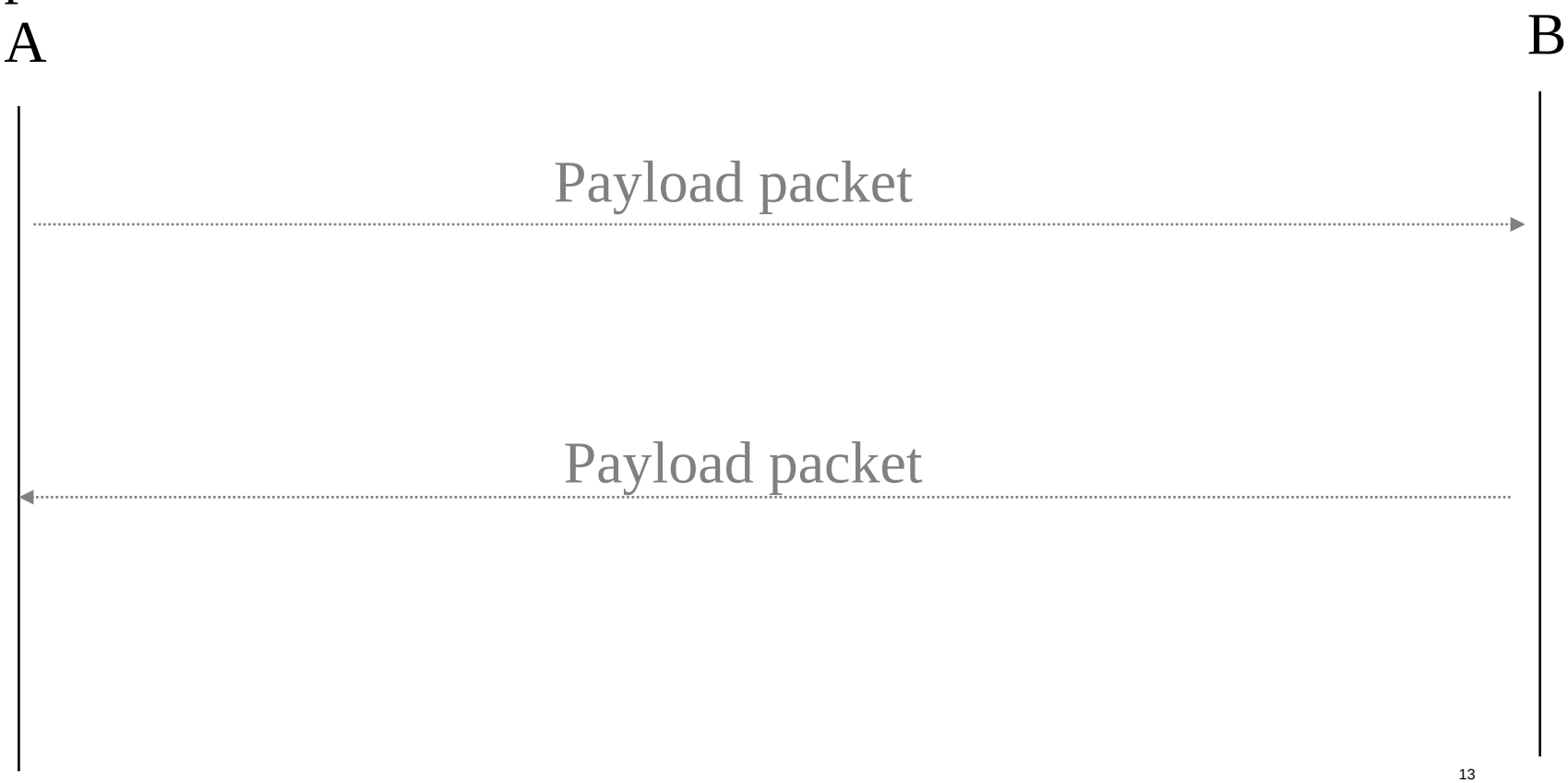
Protocol case 1 - Idle

If you are not sending or receiving payload packets, assume path is OK

A B

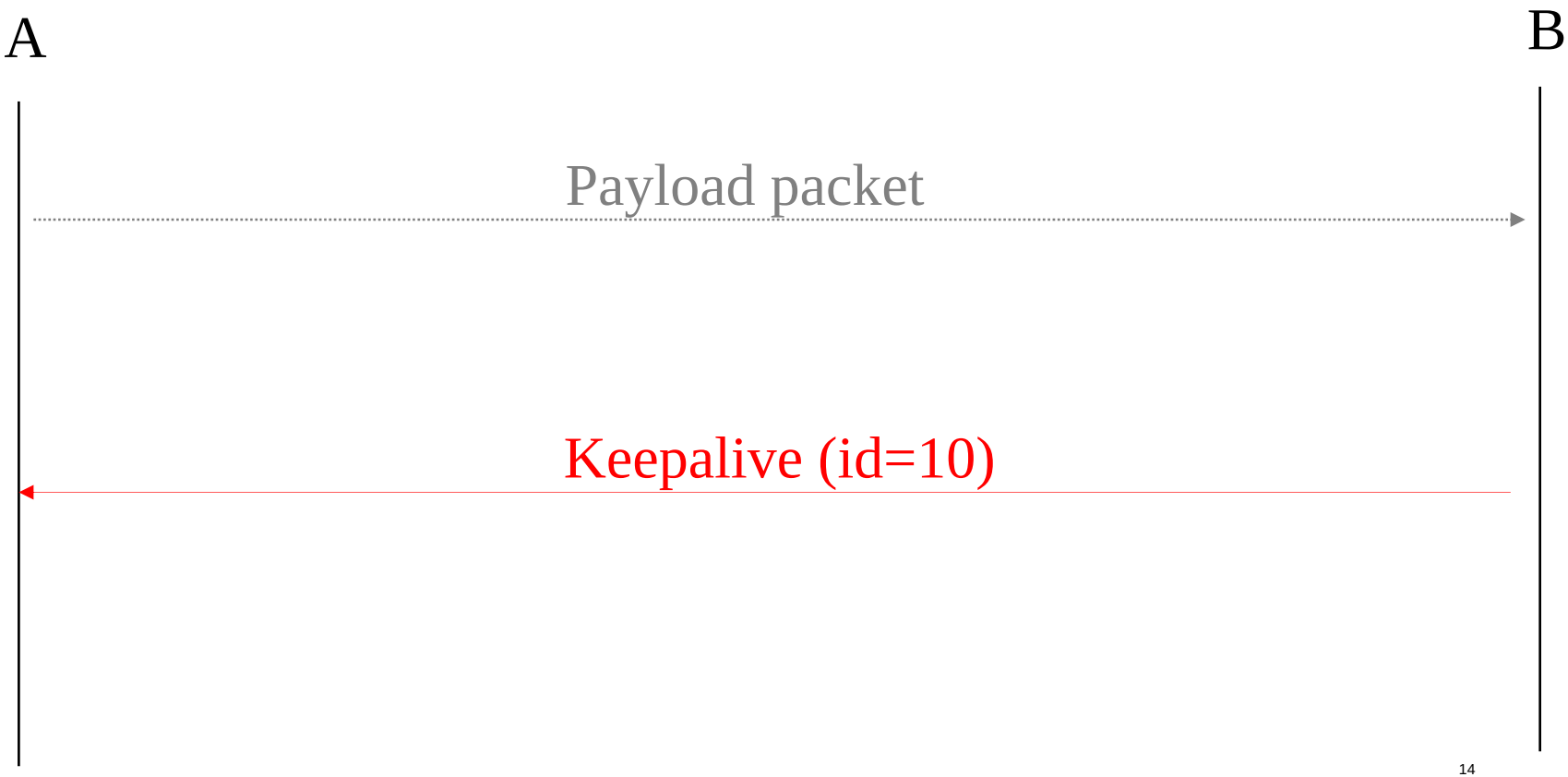
Protocol case 2 - Bidirectional Traffic

If you send and receive payload packets, assume path is OK



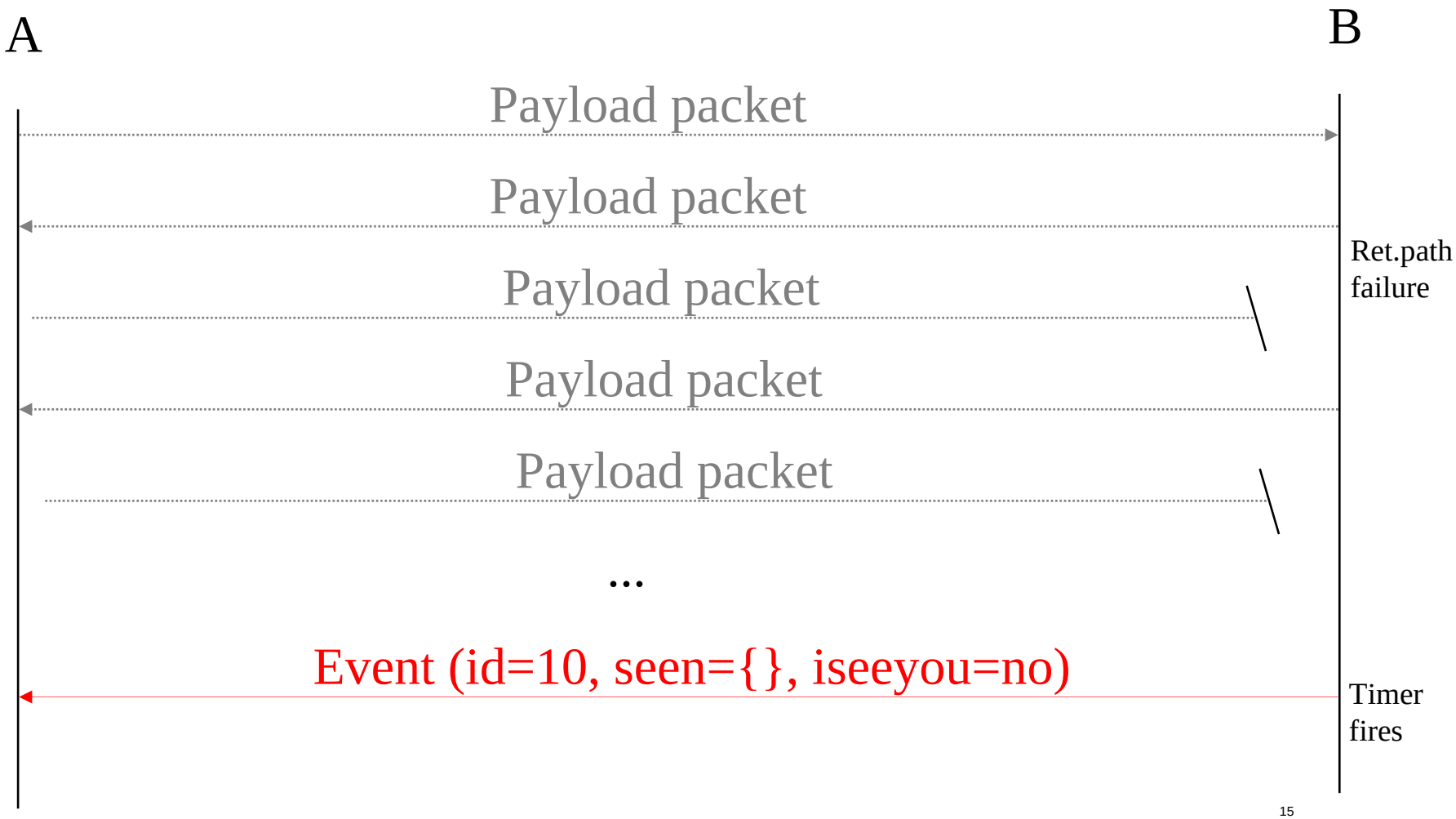
Protocol case 3 - Unidirectional Traffic

If you are receiving payload packets but not sending within t, assume path is OK but send an event message

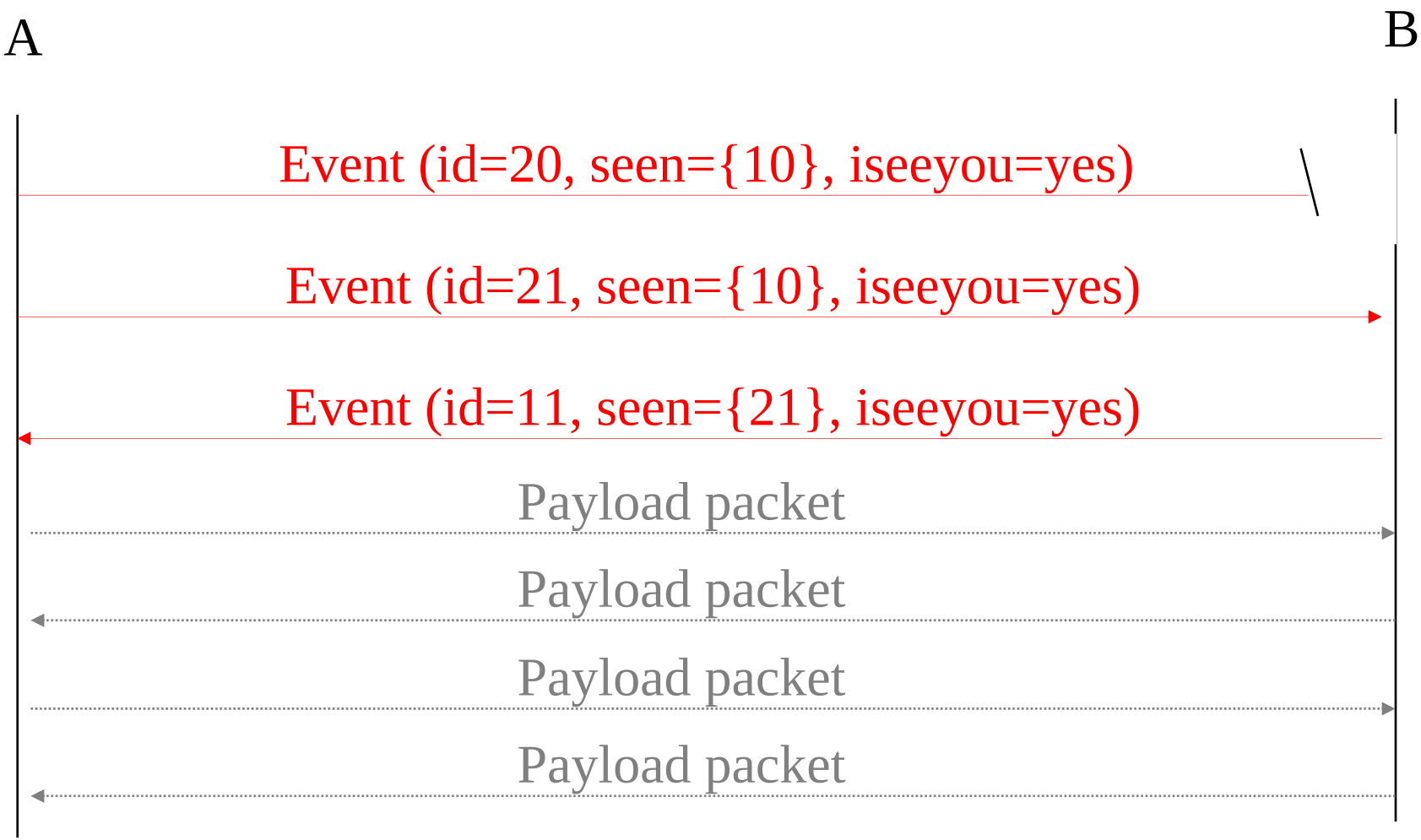


Protocol case 4 - Unidirectional Failure

If you are sending payload packets but not receiving anything, request peer to explore other return paths. Return path failure:

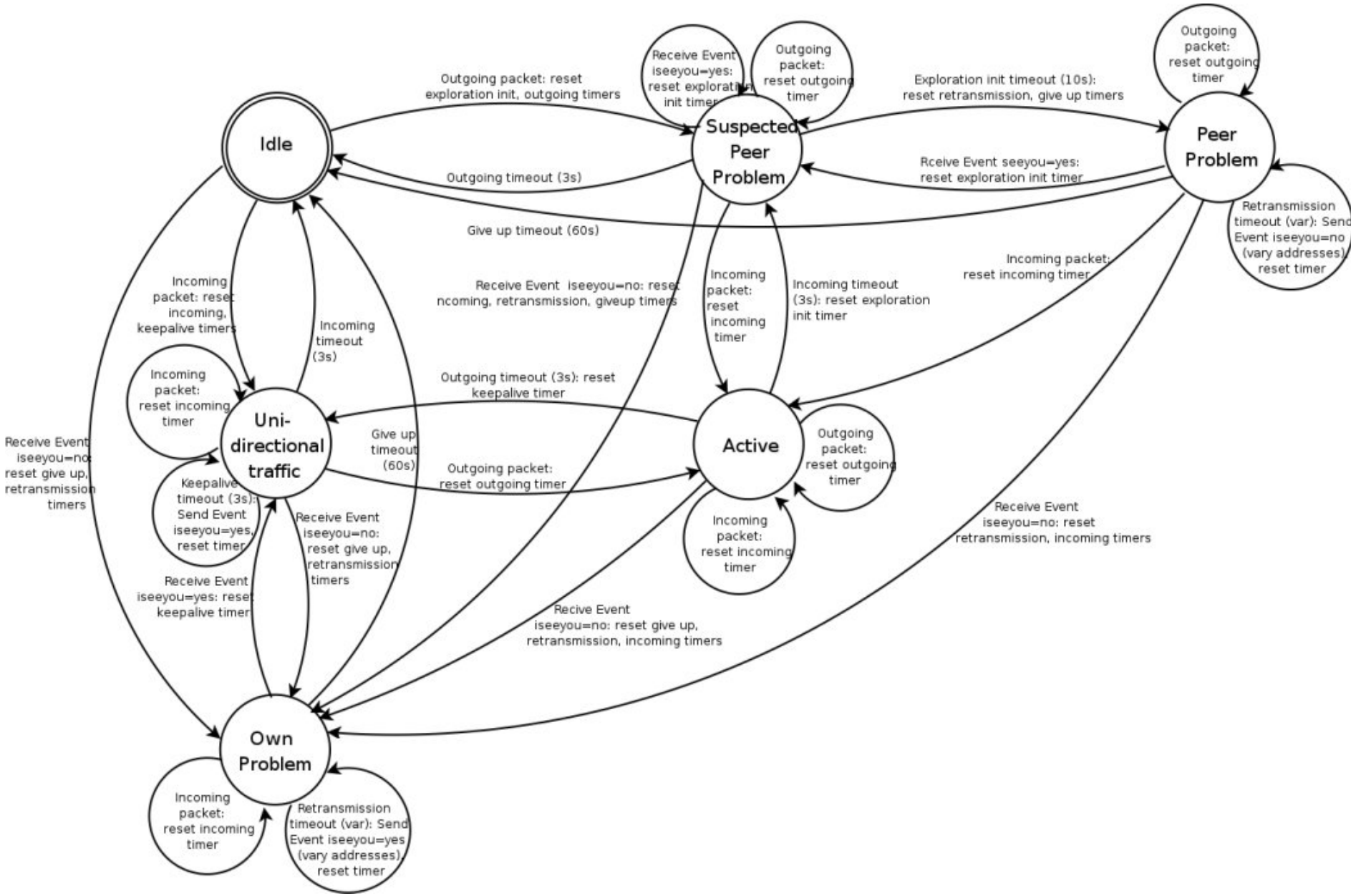


Case 4 continued



Behaviour in the General Case

- State machine in -02
- The simplified description
- The hard parts are
  - Given unidirectional connectivity, we can not use request - response
  - Stopping exploration when both parties are happy
  - Anything can happen at any time


Idle
Suspected Peer Problem
Peer Problem
Uni-directional traffic
Active
Own Problem
Outgoing packet: reset exploration init, outgoing timers
Receive Event iseeyou=yes: reset exploration init timer
Outgoing packet: reset outgoing timer
Exploration init timeout (10s): reset retransmission, give up timers
Outgoing packet: reset outgoing timer
Outgoing timeout (3s)
Rceive Event seeyou=yes: reset exploration init timer
Give up timeout (60s)
Retransmission timeout (var): Send Event iseeyou=no (vary addresses) reset timer
Incoming packet: reset incoming, keepalive timers
Incoming timeout (3s)
Receive Event iseeyou=no: reset ncoming, retransmission, giveup timers
Incoming packet: reset incoming timer
Incoming timeout (3s): reset exploration init timer
Incoming packet: reset incoming timer
Outgoing timeout (3s): reset keepalive timer
Give up timeout (60s)
Outgoing packet: reset outgoing timer
Outgoing packet: reset outgoing timer
Incoming packet: reset incoming timer
Receive Event iseeyou=no: reset give up, retransmission timers
Keepalive timeout (3s): Send Event iseeyou=yes, reset timer
Receive Event iseeyou=yes: reset keepalive timer
Receive Event iseeyou=no: reset retransmission, incoming timers
Recive Event iseeyou=no: reset give up, retransmission, incoming timers
Incoming packet: reset incoming timer
Retransmission timeout (var): Send Event iseeyou=yes (vary addresses) reset timer

The Simplified Description -- Reachability Part

keepalive-t to fire when need to send a keepalive

send-t to fire when we should have gotten something back

- RECV payload => START keepalive-t;
  STOP send-t
- RECV keepalive => STOP send-t
- SEND packet => STOP keepalive;
  START send
- TIMEOUT keepalive-t => SEND keepalive

The Simplified Description -- Exploration Part

- TIMEOUT send-t => Go to exploration;
  SEND Event iseeyou=no
- RECV Event iseeyou=no
  => Go to exploration;
  SEND Event iseeyou=yes
- RECV Event iseeyou=yes
  => Go back to normal;
  SEND Event iseeyou=yes (if necessary)

The Simplified Description -- Other

- Message identities can also confirm routability
- Optimistic - the reachability process continues even during exploration, including all timers & keepalives
- Optimistic - data packets continue to be sent to the current addresses until new addresses are confirmed (unless interface is down)
- The processes never give up (but state might be garbage collected)

Issues to Think About

- How to avoid endless yes-yes-yes… loop, i.e. how to end exploration?
- Keepalive and exploration are same or different messages?
- Processes are per context or per host?
- Timing issues vs. transport
- Issues to merge in from Iljitsch's draft

How to Stop Exploration

- Description on the slides makes the process continue forever
- One approach would be to say that a response is not sent if peer's iseeyou=yes and the peer reports seeing an event where our iseeyou=yes
- Another approach would be to use an ack message or new flag
  - But the key question is really when is this used, not so much the way it is represented

Keepalive and Exploration Integrated?

- Current approach is not
- -02 had an integrated approach
- Separation is likely cleaner
- Message formats are sufficient to express the difference, but not sure if this interacts with the previous issue
- Recommendation: keep them separate

Processes Are per Context or Host?

- Current draft does not really take any position on this
- Could explore & verify reachability in such a way that it affects all SHIM6 contexts between the two hosts
- Or could do this per context
- The latter is simpler, the former is more efficient if there are many contexts
- Iljitsch’s draft uses a per host approach

Timing Issues vs. Other Layers

- What is the time scale of SHIM6 reactions?
- Interaction with
  - Transports, TCP retransmit
  - Site and ISP routing and TE mechanisms
- In some cases we know we have problem (e.g. green light on interface card goes blank)
- In some other cases we probably should not act
  - Faster than TCP's retransmissions (e.g. not under 5 s)
  - Later than TCP or user gives up (e.g. under 30 s)

Issues to Merge in/Think about from Iljitsch's Draft

- Separation to outgoing data/other packets
- More detail on the preference values
- Also some other heuristics on selecting the most likely addresses
- Initial aggressive rate
- Formats are different