TCPCLv4 draft 07

Discussion

Victoria Pritchard, Airbus

IETF 101 - 23rd March 2018

Partial Transfers / Reactive Fragmentation

RFC4838

Reactive Fragmentation

DTN nodes sharing an edge in the DTN graph may fragment a bundle cooperatively when a bundle is only partially transferred. In this case, **the receiving bundle layer modifies the incoming bundle to indicate it is a fragment, and forwards it normally. The previous- hop sender may learn (via convergence-layer protocols, see Section 6) that only a portion of the bundle was delivered to the next hop, and send the remaining portion(s) when subsequent contacts become available** (possibly to different next-hops if routing changes). This is called reactive fragmentation because the fragmentation process occurs after an attempted transmission has taken place.

Partial Transfers / Reactive Fragmentation - draft-07 contents

- Negotiated in **Contact Header** using a **Header Extension Item**
  - Flags (CRITICAL flag means “peer node has to interpret and negotiate the reactive fragmentation capability”)
  - Type = REACTIVE_FRAGMENT
  - Length = 1 octet
  - Value = a flags field (CAN_GENERATE and CAN_RECEIVE)
- CAN_GENERATE - `the sending node is capable of generating reactively fragmented bundles`
- CAN_RECEIVE - `the sending node is capable of receiving and reassembling reactively fragmented bundles`

Partial Transfers / Reactive Fragmentation - draft-07 contents

- If sender CAN_GENERATE reactive fragments and receiver CAN_RECEIVE and pass on partial transfers in reactive fragments:
  - Sender CL gets ACKs for received data, informs BPA, which (if transfer fails) can form a smaller bundle containing the unacknowledged part, send via any CL
  - Receiver CL hands bundle data to BPA even if transfer is interrupted, BPA encapsulates as Bundle Fragment 1, and trusts that sender is creating and sending Fragment 2 (outside scope of CL)

Partial Transfers / Reactive Fragmentation Discussion

- Other combinations
  - One side omit, one side include extension → no reactive fragmentation
  - Critical/not - “peer node has to interpret and negotiate the reactive fragmentation capability”
  - CAN_GENERATE / not CAN_RECEIVE
  - Establish session but disable ACKS and reactive fragmentation?
- Is CAN_RECEIVE related to letting BPA peek at incoming bundle?
- Peers can both initiate transfers within this session
  - Is the contact header info checked for each transfer to check if the transfer in that direction can use reactive fragmentation?
  - Or does it constrain the session to one-way transfer – if the peer needs to send a bundle back, would it need to initiate a new session with contact header set up for its own requirements?

Partial Transfers / Reactive Fragmentation Discussion

- When to transfer received data from CL to BPA?
  - Since draft 1, BPA gets to inspect a bundle before it is fully received, and can signal to the CL to refuse the bundle, to stop the sender transmitting any more of it
  - 5.3.5 "`A XFER_REFUSE can also be used after the bundle header or any bundle data is inspected by an agent and determined to be unacceptable.`" + mentioned in other places
- Threshold on amount received before doing reactive fragmentation?
  - e.g. sender says if 50% was received, I'll create a fragment containing the rest. Receiver needs to agree - 50% was received so I'll hold on to it and wait for a fragment with the rest. Should the threshold be configurable? in BPA config? How do you exchange/agree on this threshold?

Signals between CL and BPA

BPA ↔ CL

→ Attempt session

→ Shutdown session

← Session started

- TCP connection open

← Session established

- TCPCL session – i.e. contact headers exchanged, ready to use

← Session shutdown

← Session failed

Signals between CL and BPA - sending a bundle

BPA ↔ CL

→ **Begin transmission** - here's a bundle

← **Transmission availability** - session open and idle

- Do we get same information from Session Established and Transmission Success/Failure?

← **Transmission success** - bundle fully transferred

← **Transmission intermediate progress**

- `at the granularity of each transferred segment` - number of bytes acknowledged?

← **Transmission Failure**

- Why not send number of bytes sent so far here?
- "`The TCPCL supports positive indication of certain reasons for bundle transmission failure`" - are the reason codes to be sent to the BPA too?

Signals between CL and BPA - receiving a bundle

BPA ↔ CL

← Reception intermediate progress

- `at the granularity of each transferred segment`
- Does this mean data from each segment is passed up to BPA as it arrives, or is it a segment count or byte count?
- `Intermediate reception indication allows a BP agent the chance to inspect bundle header contents before the entire bundle is available, and thus supports the "Reception Interruption" capability.`

→ Interrupt reception

- Send a XFER_REFUSE to stop transfer before it has completed (see Reception intermediate progress)

← Reception success - bundle fully received

← Reception Failure

Session Shutdown

- When can you use SHUTDOWN message?
  - To refuse session setup (but after contact header exchange)
  - Not while a TCPCL message is currently being sent (but can close the TCP connection in this case)
  - Before in-progress transfers have completed, i.e. after XFER_INIT or XFER_SEGMENT have completed transmission
  - After an idle period (where only keepalives are being sent)
  - Up to the implementation
- Can include a reason code
- Receiver of SHUTDOWN SHOULD send a SHUTDOWN in reply

Session Shutdown

- After SENDING a SHUTDOWN
  - Do not initiate any new transfers
  - Not forbidden to accept any new transfers
  - MAY immediately shutdown TCP connection
    - If closing due to idle, not an issue
    - If not idle, sender of the SHUTDOWN may still need to ACK segments from the peer

Session Shutdown

- After RECEIVING a SHUTDOWN
  - SHOULD send a SHUTDOWN in reply (then see previous slide)
  - Don’t accept new transfers
    - Ignore? Refuse? Reject? Send a(nother) SHUTDOWN?
  - Not forbidden to initiate new transfers here (unless a SHUTDOWN is sent)
  - SHOULD send all ACKs before closing the TCP connection

Session Shutdown - Unclean

- Unclean SHUTDOWN
  - Draft defines this as closing TCP connection immediately after sending SHUTDOWN
  - If session is idle, is that an unclean SHUTDOWN?
  - Is it more accurate that an unclean SHUTDOWN is when you close the TCP connection before a transfer is finished (since this can be done mid-segment or before the final ack)?

Session Shutdown - Unclean

- When performing an unclean shutdown, a receiving node SHOULD acknowledge all received data segments before closing the TCP connection.
  - Does **receiving node** mean the **receiving side of a transfer**?
  - SHOULD send all ACKs before closing TCP connection, i.e. SHOULD NOT perform unclean SHUTDOWN?
- When performing an unclean shutodwn, a transmitting node SHALL treat either sending or receiving a SHUTDOWN message (i.e. before the final acknowledgment) as a failure of the transfer.
  - If “performing” an unclean SHUTDOWN (I.e. TCP close), why mention send/receive of a SHUTDOWN?
  - Also numerous places where draft says to continue sending ACKs, finish in-progress transfers, so after a SHUTDOWN the transfer can still complete successfully
  - Make it more general? If the TCP connection is closed, before the final acknowledgment has been sent/received, this is a failure of the transfer.

Session Shutdown

- Reason codes
  - If a transfer is in progress, only relevant code is Resource Exhaustion, but the draft allows transfers to finish
- Reconnection delay
  - Sending 0 means “never reconnect” - is this wise? Can it be un-done?

Thank you
Any questions?