© 2019 Mellanox Technologies

# UCP problems with protocols/endpoints

- Complicated logic for endpoint lanes selection
- Complicated logic for threshold selection
  - Possible inconsistencies between protocols (e.g max_short vs. rndv_thresh)
- For non-inline case: many data-path checks for message size, datatype, memory type
- If endpoint changes configuration, send request should be re-initialized
  - Happens with client-server applications
- Packing functions have side effects on request state
  - Happens with ugni
- Can’t reuse multi-rail, memory type handling code between RNDV and RMA

© 2019 Mellanox Technologies 

# Solution approach

- Separate protocol definition from protocol selection engine
  - Generate thresholds in a protocol-agnostic way
- Separate lane requirements definition from lane selection engine
- Keep ep config index in ucp_request_t to detect need to re-initialize
- Reuse protocols for different operations
  - the data fetch phase of RNDV protocol is same as ucp_get()

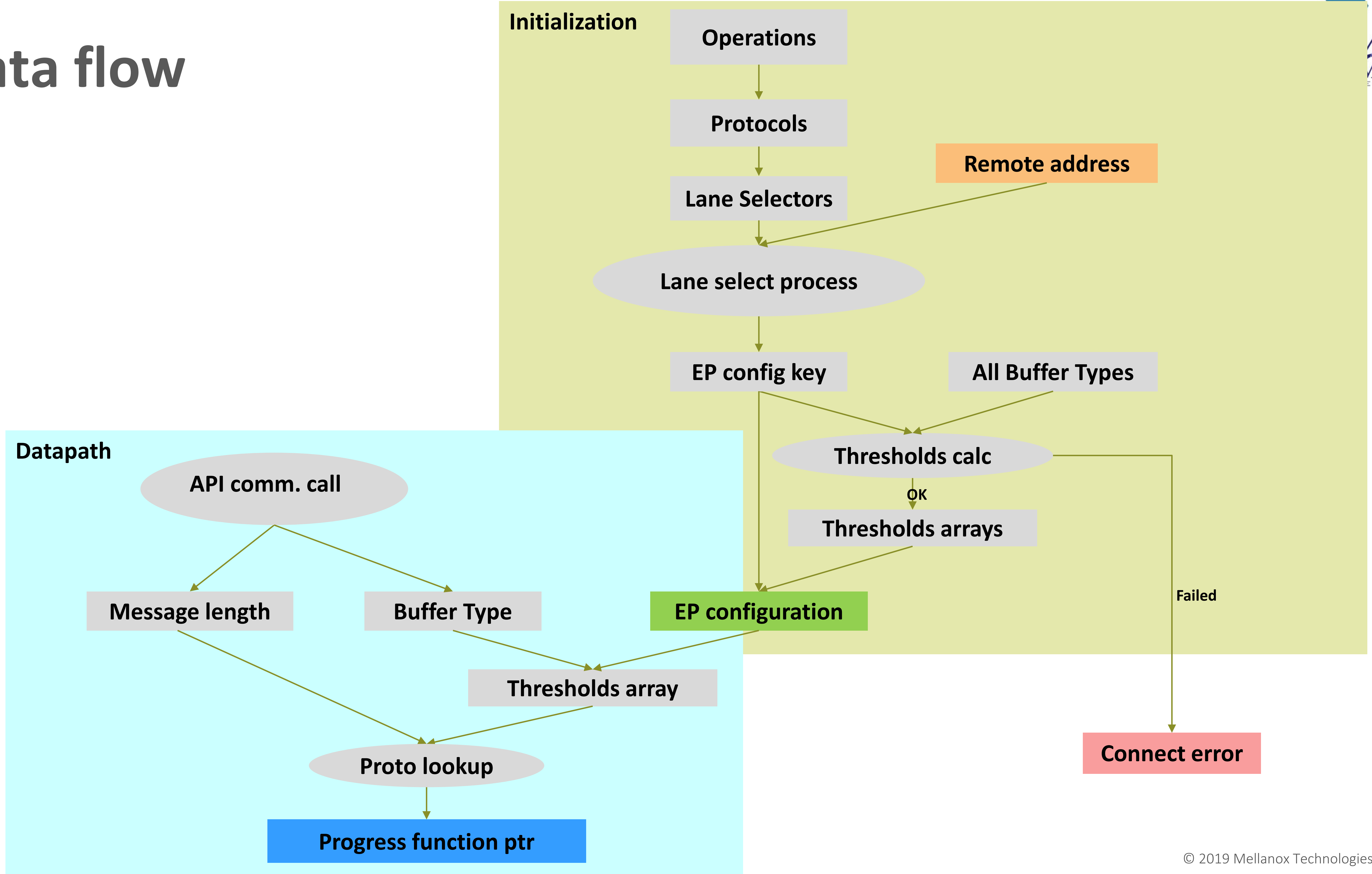
© 2019 Mellanox Technologies

# Object relationships

- Operation
  - High level API or internal operation which could be executed by one or more “protocol”.
  - For example: ucp_tag_send(), ucp_get(), rdnv_rts_send
- Protocol
  - Defines method to implement one or more “operation”, by a UCT progress callback
  - Defines the “lane selectors”, to have lanes required for its execution
  - Exposes performance estimation and min/max message size, per “buffer type”
  - For example: eager/bcopy/multi, eager/zcopy/single
- Lane selector
  - Defines criteria for selecting a group of lanes for an endpoint
  - Defines a function which checks a potential <local,remote> combination and returns if it’s valid and its score+priority
    - The function is called repeatedly to select multiple lanes until no more are found
  - For example: TAG, AM_BCOPY_BW, RMA_BW
- Buffer type
  - A tuple of (datatype, memory type, sg_count) which defines memory layout properties
  - Will add locality information in the future (for GPU/NUMA awareness)
  - Protocol performance is always relative to a buffer type

© 2019 Mellanox Technologies

# Data flow




© 2019 Mellanox Technologies

# Endpoint initialization

1. According to context features, define the set of required operations
2. According to (1) , and in the future also by wire version, define set of protocols to init
3. According to (2), get the set of all lane selectors
4. Iterate over all remote/local combinations and select endpoint lanes. Some could be empty.
5. For each possible operations and buffer type, initialize the thresholds array which select protocol for every possible message size. If a protocol could not find lanes, don’t use it.
6. If some operation/buffer type could not fill the protocol selection, show an error that we don’t have enough transport lanes to connect.

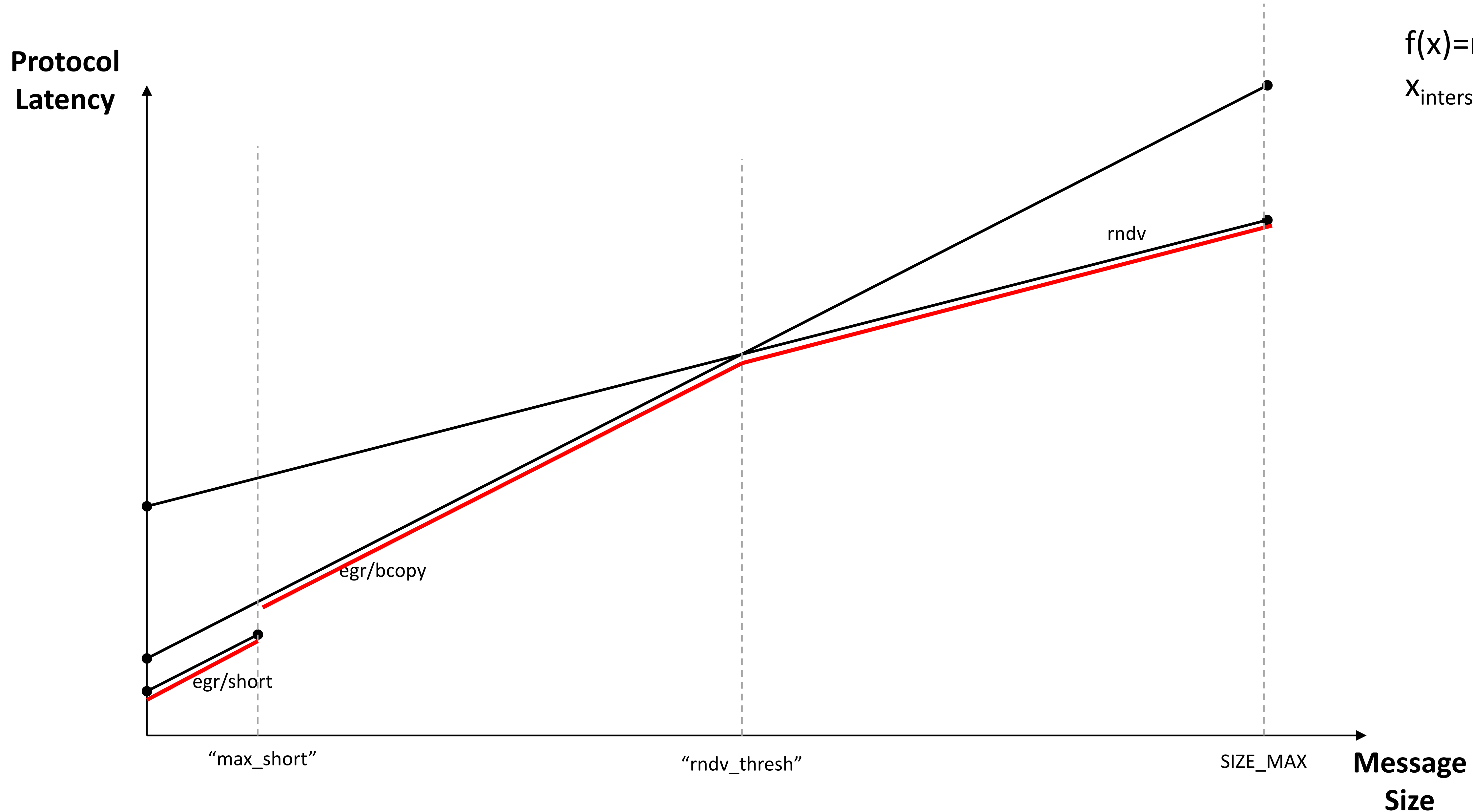
© 2019 Mellanox Technologies

# Protocol threshold selection

- The performance of a protocol on a buffer type represented as linear function of “time to send” whose argument is message size, e.g T(message_size) = c + m*message_size
- Find the best protocol for every “interval” by walking on the intersections between the linear functions of all available protocols. Continue doing so until all the range 0..SIZE_MAX is covered.
- Save the selection result in an array of <max_message_size, proto>. The max_message_size of the last array element is always SIZE_MAX.
- Protocols which are not used will not appear in the array

© 2019 Mellanox Technologies

# Protocol threshold calculation




$f(x)=m*x+c$ ➔

$x_{intersect} = (c_2-c_1)/(m_1-m_2)$

# Runtime protocol selection

- Construct a hash key for buffer type lookup – by combining datatype, detected memory type, and sg_count (for IOV type)
- Perform a hash lookup from this key to protocol selection array
  - Should be faster than the multiple checks and branches we have today
  - Contig/host type is fast path lookup without hash
- Walk on the thresholds array and find the first entry whene our buffer size <= max_msg_length.
- Use the protocol in the given entry by executing its progress function

© 2019 Mellanox Technologies

# Thank You