HEADER COMPRESSION DESIGN TEAM

MIKE BISHOP

ALAN FRINDELL

BUCK KRASIC

ROBERTO PEON

MARTIN THOMSON

DMITRI TIKHONOV

LATENCY IS THE ENEMY
(AND POOR COMPRESSION IS LATENCY)

- Head-of-line blocking
  - Reordering
    - Particularly from loss, but also network and even internal
    - Always impacts the current stream, can impact other streams
  - Data loss
    - Packet drops in combination with RST_STREAM (i.e. never retransmitted)
- Bandwidth limitations
  - Fit more requests into allowed bytes

THE MISSION

Static HPACK suffers from poor compression efficiency

Serialized HPACK suffers from head-of-line blocking

THE MISSION

Static HPACK suffers from poor compression efficiency

Serialized HPACK suffers from head-of-line blocking

Efficiency + Blocking Avoidance = Unicorn

OPERATING CONDITIONS

- Reordering is common
  - Network reordering varies widely across networks
  - Loss and retransmission is fundamentally a reordering event
  - Multi-threaded implementations may induce reordering internally
- Many connections experience no loss
  - Not so many that we can discount this
  - Not so few that we should penalize the majority for the minority's crummy link
- Request cancellations occur with some frequency
  - Only ~0.8% of **requests** are reset (Facebook)
  - ~51% of **connections** experience at least one reset (Akamai)

HOW TO HANDLE REORDERING: BLOCKING

FULL ORDERING

- Risks false sharing in head-of-line blocking
  - Single packet lost from this stream blocks headers on all streams
- Worst possible HOLB rates

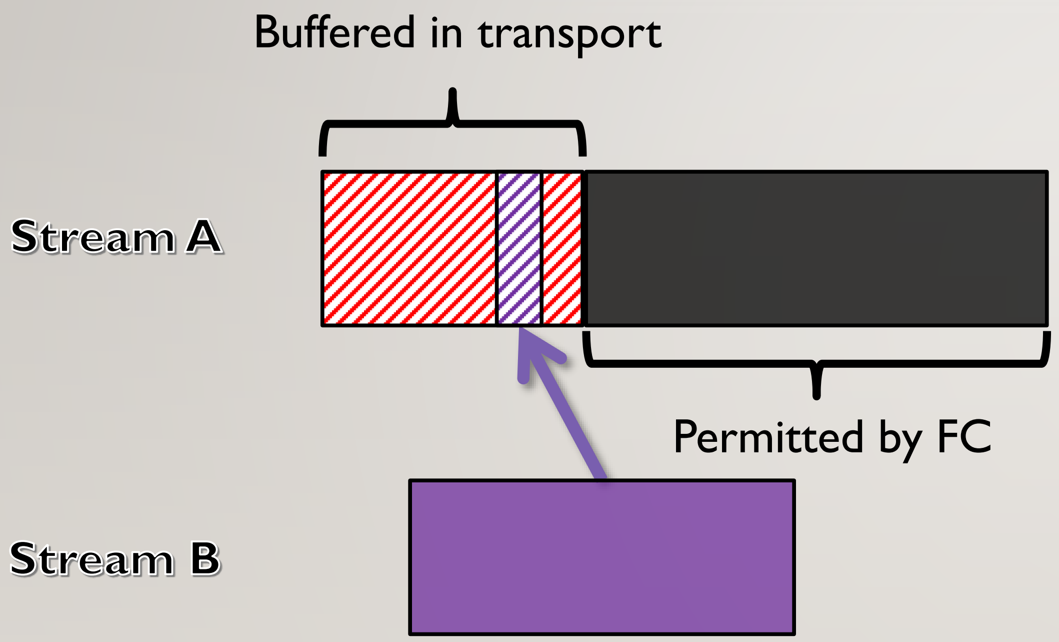
OPTIMISTIC CONCURRENCY

- Assumes state has arrived
  - Block only if necessary state is missing
- Uses flow control to provide back-pressure and control memory consumption
- Risks deadlocks

NEVER RISK BLOCKING

- Robustness
  - Avoids risks of deadlock, memory consumption, etc.
- Efficiency suffers noticeably
  - Must add headers to table at least 1 RTT in advance of using them, or else send them multiple times during first RTT of use

HOW TO DEADLOCK



- Interpretation of Stream B depends on data from Stream A
- Flow control prevents data on Stream A from being sent
- Lack of progress on Stream B prevents new flow control credit from being issued to Stream A

HOW TO NOT DEADLOCK

Don’t Do That!

- **Problem:** Can all application protocols avoid this all the time?
- **Problem:** Really hurts compression performance

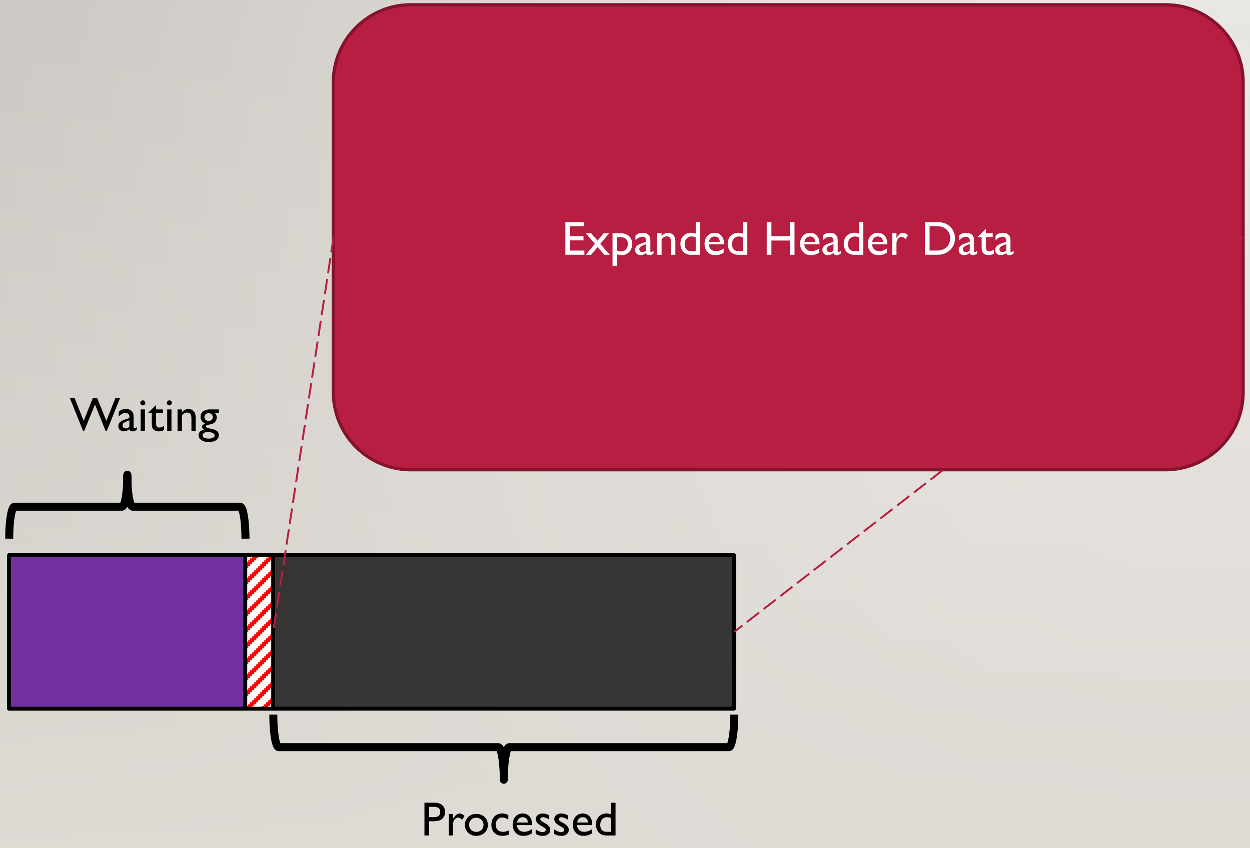
Prioritization Between Streams

- Ensure Stream A makes progress with any new flow control credit that becomes available
- **Problem:** Priorities are currently:
  - Purely advisory => optional
  - Internal to the transport implementation’s design

Consume Flow Control Sooner

- Flow control consumed on write completion, not on transmission
- Application responsible to make sure data written to A before beginning write to B
- **Problem:** Application-level retransmits

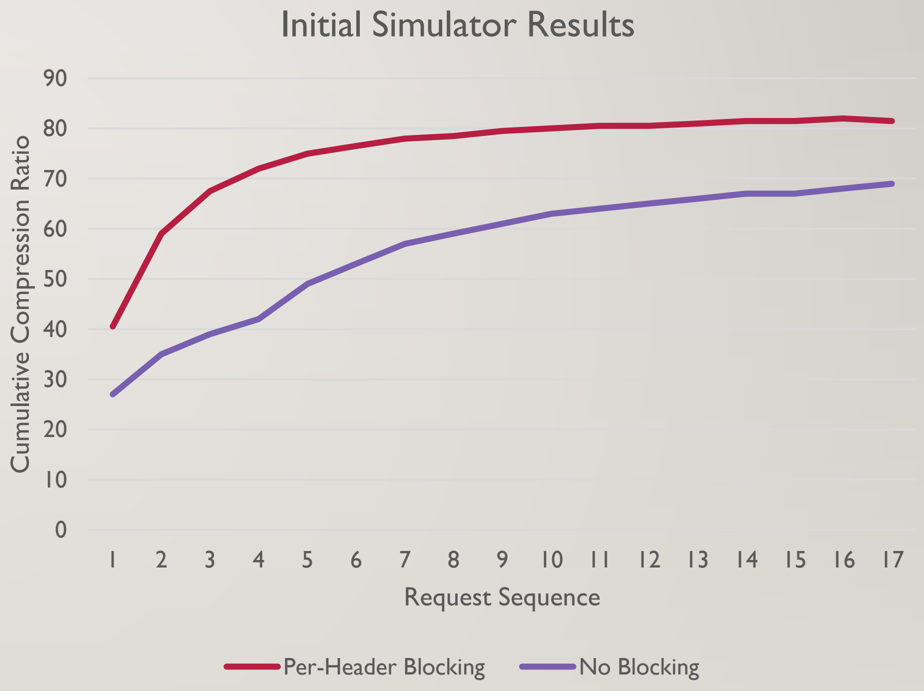
LIMITING MEMORY CONSUMPTION



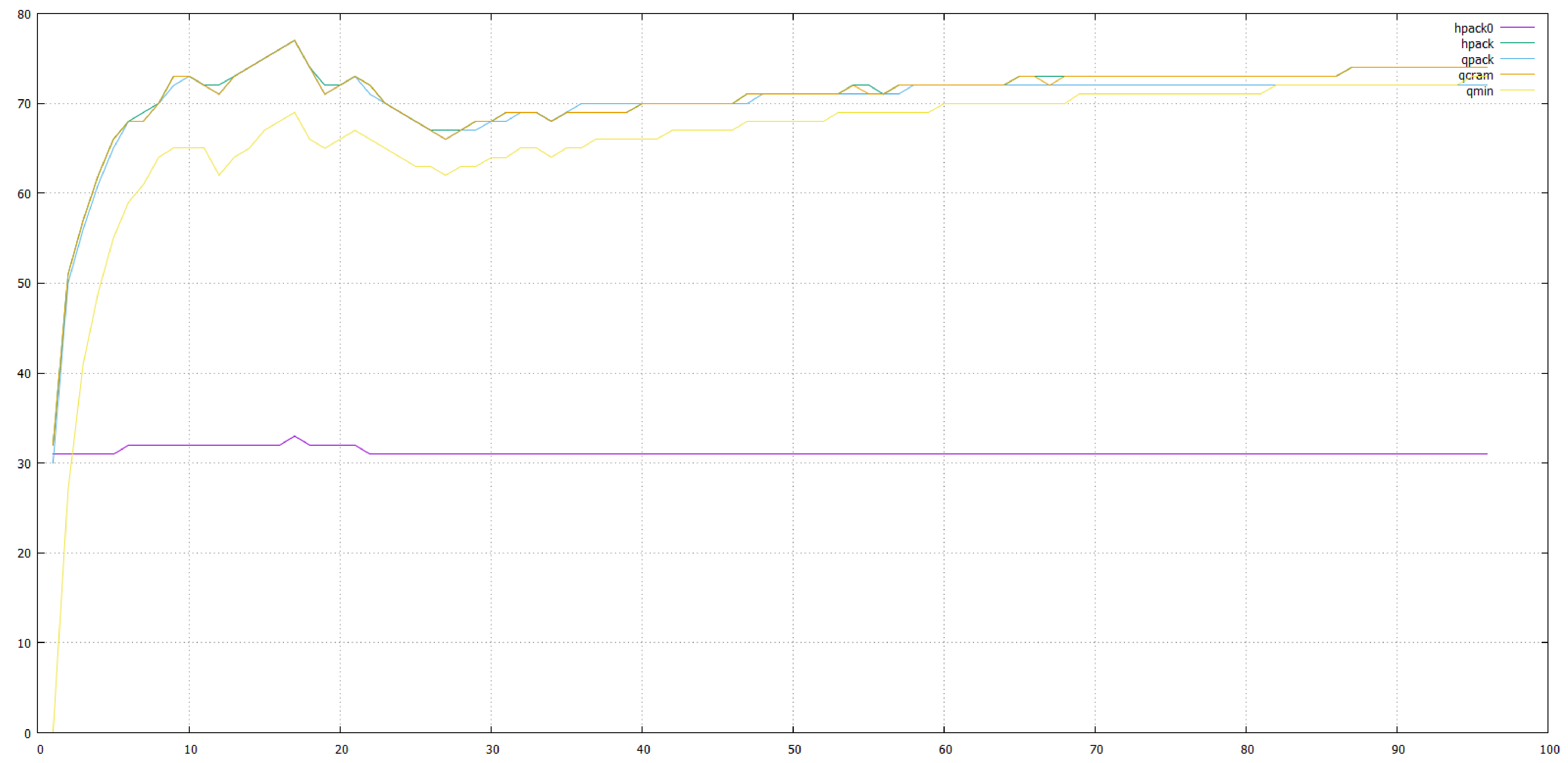
- Discovering a blocking reference mid-frame means you already have uncompressed data in memory
- Suggestion: Don't begin reading a frame until you have all necessary state to finish
  - Uses flow control for back pressure
  - Requires frame preface describing encoder state
- Separate from blocking on missing data

SIMULATOR RESULTS

- Allowing blocking means carefully balancing ways to avoid deadlocks
- Noticeable compression gains early in the connection
  - No simulator yet for per-set blocking
- No data yet on exactly how this translates to latency



SIMULATOR RESULTS: LONGER SESSION



IN SEARCH OF GUIDANCE

- Should blocking be allowed?
- If so…
  - What is the scope of blocking?
  - Should blocking be configurable?
  - Is it acceptable to leave compressed data unread from the transport while blocked?

HOW TO BUILD CONTROL STREAMS

MANY CONTROL STREAMS

- Mitigates the impact of loss between unrelated entries
- Requires transport features to guarantee no deadlocks

SINGLE CONTROL STREAM

- Simplifies deadlock avoidance
- Efficiency suffers in the presence of loss

MINIMIZE THE CONTROL STREAM

- Simplifies common case
- After aborted stream, re-writes critical data on control stream

HOW TO TRACK DATA

DATA PER HEADER

- Each header is individually added, referenced, and deleted
- DT has largely eliminated due to memory/CPU overhead

CHECKPOINTS

- Groups of header entries
- Track which/how many checkpoints reference entry
- When all referencing checkpoints are gone, header is removed

ROTATING WINDOW

- Headers added in sequence (HPACK-style)
- When table size reaches limit, old entries roll off

WHERE ON THE SPECTRUM ARE WE?

- QPACK:
  - -06 and previous: Multiple control streams, blocking and data per header
  - -07: Multiple control streams, optimistic concurrency, checkpoints
- QMIN:
  - Single control stream, never blocks, checkpoints
- QCRAM:
  - Avoids control stream, optimistic concurrency, rotating window

REACHING FOR CONSENSUS

- Transport-level ACKs, even if exposed, only tell us about arrival, not processing
  - All three proposals now incorporate application-level acknowledgements of updates
- Per-table-entry state tracking is too onerous
  - Too much memory to maintain all relevant state
  - Too much CPU to update all relevant entries on state changes
  - QPACK-07 adopts QMIN-style checkpoints instead

MOVING FORWARD

- Need more data to settle questions
  - What is the latency impact of compression efficiency gaps?
  - What is the latency impact of losses under optimistic concurrency?
- Simulation/implementation updates in progress
  - Alan implementing QPACK-07
  - Buck implementing QCRAM-03
- Need input from implementers about tolerance of optional features