Transport-Related Protocol-Design Issues

WG Chairs Training Lunch

IETF-73, Minneapolis, MN, USA

Magnus Westerlund & Lars Eggert

November 19, 2008

Outline

- Background
- Overall Recommendation
- Make No Assumptions
- Common Transport-Related Protocol-Design Issues
- How to Use a Standard Transport
- Where to Go for Design Help

Background

- The Internet is a shared resource for end-to-end communication
- It does not provide any network-side resource allocation and sharing mechanisms
- These functions are provided through the common, distributed, end-to-end transport algorithms executing in each end system
  - Congestion control (or at the very least overload prevention)
  - Approximate fairness (or at the very least starvation prevention)
- It is the duty of each application and application-level protocol to implement the necessary algorithms to be a well-behaving consumer of Internet resources

Overall Recommendation

- Use a standard transport protocol that already provides these required mechanisms – TCP, SCTP, DCCP
  - You're mostly done!
  - (Note the absence of UDP – it's NOT a transport protocol)
- However, some applications are special
  - And many more *think* they're special
- So they need to implement some reasonable mechanism that lets them be a responsible consumer of Internet capacity
- This is a difficult problem – there are no easy solutions
  - Otherwise, we'd have much simpler transport protocols already

Make No Assumptions

- The Internet is a diverse place
  - Link rate diversity from <1Kbps up to 40 Gbps
  - One-way delay from <1 μs to several seconds
  - Multiplexing diversity from 1 flow/path to infinity
  - Supported packet size from 68 bytes to > 9K bytes
  - Packet loss, reordering, duplication
  - Path characteristics change dynamically over short timescales
  - No capacity reservations, no special treatment of some traffic
  - Communication resources are shared & there is other traffic
- Make no assumptions on any of the above – an Internet protocol must operate correctly anywhere on the Internet
  - Otherwise, it only works for walled gardens

“But I Have a Walled Garden…”

- So, that let’s you make some assumptions
  - And you will need to make them VERY explicit in your document
- However, your protocol will still need mechanisms that let it handle failure conditions gracefully
  - Even walled gardens can have failures
- And it needs some fail-safes in case folks end up running it on the big-I Internet anyway
  - Ask yourself: “What happens if this ends up in Linux?”
- At this point, “real” transport protocols usually end up looking more and more attractive…

Example: Quality of Service

- Relying on QoS mechanisms restricts your protocol to controlled environments where the deployment and configuration of those mechanisms can be enforced
- Examples
  - Differentiated forwarding (DiffServ) RFC 2475
  - Capacity reservations (IntServ or static) RFC 1663, 2210-12
- Relying on QoS is usually only appropriate for protocols that are (only) useful for such controlled environments
  - e.g., less-than-best-effort PHB [LEPHB] is not going to help with the Internet-wide P2P flood

Common Transport-Related Issues

- Designing from scratch – what do you need to worry about
  - Congestion control
  - Path MTU & message sizes
  - Communication Reliability & Integrity
  - Ordering & Duplication detection
  - Multiplexing
  - Middleboxes
  - Performance
  - Multicast
  - Tunneling
  - Resource Discovery
- For UDP-based designs, see RFC 5405
- Once you've dealt with these, your homegrown scheme is unlikely to be much simpler than a standard transport

Congestion Control

- Congestion control = prevent overload + be reasonably fair
- Preventing overload requires detecting congestion and responding to it by reducing traffic load
  - Need mechanism to detect congestion
  - Need algorithm to determine safe transmission rate
- Being reasonably fair is harder
  - Don’t have a crisp definition of fairness
  - Typical approach is to “not consume more bandwidth than a TCP connection would under similar network conditions”
- Building efficient mechanisms for this correctly is very hard
  - Control theory: stability, oscillations, reaction time, effectiveness
  - People get their PhDs doing these

Congestion Control (II)

- Mechanism must function correctly over the full range of possible network conditions
- There are some standard building blocks available
  - AIMD (TCP) RFC 2581
  - Equation-based (TFRC) RFC 5348
  - Lock-step (marginally useful)

Path MTU & Message Sizes

- Persistent IP-level fragmentation makes Internet communication brittle and inefficient
  - IPv4 ID field mis-assocation RFC 4963
  - Middleboxes commonly discard IP fragments
  - Allows for security vunerability
- Protocols must adapt their message sizes to the MTU a given path can natively support
  - Or limit themselves to the minimum MTU, but that’s inefficient, too (header overhead)
- Do path maximum transmission unit (MTU) discovery
  - But ICMP can no longer be relied upon
  - Packetization-Layer Path MTU Discovery (RFC 4821) is a solution, but requires hooks inside the protocol to work

Communication Reliability

- Different types of reliability may be needed
  - Reliable session establishment and teardown
  - Reliable protocol configuration
  - Reliable data/message transport
- Must design handshakes & acknowledgements
  - Requires state machines & sequence numbers
  - Separate packet accounting and payload
  - Retransmission can result in uncertainty about which packet is being ACKed
  - Need to backoff retransmission timers
- State management
  - Creation
  - Cleanup
  - Handling reboots/restart

Integrity

- Remember: not all link layers have checksums
- TCP, UDP and DCCP use a ones' complement checksum
  - Can detect errors introduced in between the end-points, and things that were missed by link layer checksums
  - Quite weak, 16 bit ones-complement sum
- Protocols that require stronger protection from errors should use stronger integrity protection inside the protocol
  - Especially if lacking synchronization point in protocol messages
- SCTP uses a 32-bit CRC
  - OK for most uses
- Transfer of larger data objects over any transport should be verified

Multiplexing

- TCP, SCTP, DCCP and UDP all have 16-bit ports
  - Multiplexing is the main purpose
  - DCCP also has service codes
- Port number range is quite limited
  - A protocol should not use more than a single port number
  - May require additional protocol multiplexing points for functionality, security, etc.
  - Don't forgett versioning of the protocol
  - Dynamic server ports may be possible to use through DNS SRV RRs, mDNS, LLMNR (RFC 4795) or other mechanism
- Define what happens if the desired port isn't available
  - Complete failure
  - Or something the protocol can deal with

NAT and Middlebox Traversal

- Middleboxes are problematic
- Basically: enforce directionality on communication establishment – client/server paradigm OK, others less so
  - Complex communication establishment procedures
  - Relays, coordination, etc.
  - There are some building blocks available (STUN/TURN/ICE)
  - Also consider address family translations (IPv6 to IPv4)
- Unless you really aren't playing nice with middleboxes:
  - Structure the protocol design for this from the start!

NAT and Middlebox Traversal (II)

- Another headache: middlebox state silently times out when a session has been idle for a while
  - There are no rules for how soon this happens
- Results in communication failures when an application resumes transmission
- Popular countermeasure: keep-alives
  - “send dummy traffic when there’s nothing real to send”
  - incredibly wasteful (esp. on battery operated devices)
  - isn’t guaranteed to keep the session alive, either
  - only sensible where failure is very, very bad (cf. FEC)
- Better countermeasure: robust session handling
  - Good protocols recover from communication failures anyway
  - failures due to middleboxes are like any other failure

Performance

- Donald Knuth: “Premature optimization is the root of all evil”
  - especially when based on guesses
  - Need clear requirements based on analysis
- The performance will heavily depend on the underlying network path
- Tuning performance of existing protocols is possible, e.g.
  - For example: TCP can be tuned to not wait for additional data after a socket writes by turning off Nagle
  - However, there is no reliable correlation between application writes and segment boundaries in TCP

Multicast

- Multicast is difficult due to the:
  - Simultaneous path diversity between sender and multiple receivers
  - The fact that packet losses will only affect sub-groups of the receivers
  - Feedback Implosion
  - Trust issues for the feedback
- The Reliable Multicast Transport WG has made a number of building blocks and a few protocol instantiations for different usages
- No general recommendation come talk to people who worked on this

Designing Protocols that Tunnel

- Tunnels are virtual links – you're designing a link layer on top of IP!
- Make it look like a real link layer
- Need to think about
  - Not breaking path MTU discovery (DF bit)
  - Fragmentation due to encapsulation
  - ECN
  - DiffServ
  - Handling and translating error messages for problems in the tunnel
- draft-touch-intarea-tunnels-00

Use a Standard Transport!

(You’re not rolling your own crypto, either)

Which Standard Transport Protocol?

- Choices & trade-offs
  - TCP: Reliable byte stream, head-of-line blocking
  - SCTP: Reliable, message-based, multi-homed, NAT issues
  - SCTP-PR: Partially reliable, otherwise as SCTP
  - DCCP: Unreliable, connection-oriented, NAT issues
- UDP is not a real transport protocol
  - It’s IP with port multiplexing and checksums
    (also, UDP-Lite for partial checksum coverage)
  - Lacks congestion control, MTU discovery, etc.

Issues with Standard Transports

- Having picked a standard transport, you’re mostly done
- But some things still deserve to be specified
  - TCP connection close determines who keeps TIME-WAIT state
  - Number of simultaneous connections in use
  - Source port randomization
  - Well-known port vs. service discovery protocol
  - Message delimitation
  - Accepting Transport Protocl behaviors and design with it in mind

Where to Go for Design Help

- Get your design reviewed early and often
  - getting the end-to-end aspects of a protocol right isn't something that can be done after the fact
  - we don't like late surprises either
- Contact the transport directorate (tsv-dir@ietf.org)
- Ask the ADs for a transport advisor for your WG
- If you are building on top of UDP, read RFC 5405
- If you are doing Multicast talk to RMT WG

Use a Standard Transport!

(You’re not rolling your own crypto, either)

References

- UDP: RFC 768
- UDP-Lite: RFC 3828
- TCP: RFC 793, RFC 4614 (Roadmap)
- SCTP: RFC 3268 (Introduction), RFC 4960
- SCTP-PR: RFC 3578
- DCCP: RFC 4340
- [LEPHB]: http://tools.ietf.org/html/draft-bless-diffserv-le-phb-00
- [mDNS]: http://www.multicastdns.org/
- STUN: RFC 5389
- TURN: http://tools.ietf.org/wg/behave/draft-ietf-behave-turn/
- ICE: http://tools.ietf.org/wg/mmusic/draft-ietf-mmusic-ice/