The Transport Layer
Principles of Reliable Data Delivery

Jasleen Kaur
March 3, 2016

Transport Layer Protocols & Services

Outline

- Fundamental transport layer services
  - Multiplexing/Demultiplexing
  - Error detection
  - Reliable data delivery
  - Pipelining
  - Flow control
  - Congestion control

- Service implementation in Internet transport protocols
  - UDP
  - TCP
**Fundamental Transport Layer Services**

**Principles of reliable data transfer**

- **Goal:** Provide a reliable channel abstraction
  - The characteristics of the underlying channel will determine the complexity of providing reliable communications
- **Issues:** State required at sender and receiver and number of control messages exchanged

---

**Reliable Data Transfer**

**Programming interfaces**

- Called "from above" by the application. Application passes in data to be delivered to receiving application
- Called by rdt to deliver data to application
- Called by rdt to transfer packet over unreliable channel to receiver
- Called when packet arrives on receive-side of channel
Reliable Data Transfer

Protocol specification method

- Use finite state machines to specify sender and receiver algorithms
  - When in a given state, the next state (and actions) are uniquely determined by the next event

Reliable Data Transfer Protocol 1.0

Reliable transfer over a reliable channel

- The underlying channel is assumed to be perfectly reliable
  - No bit errors
  - No loss of packets

Sender state machine

- `wait for call from above`
- `rdt_send(data)`
- `make_pkt(pkt, data)`
- `udt_send(pkt)`

Receiver state machine

- `wait for call from below`
- `rdt_rcv(pkt)`
- `extract(pkt, data)`
- `deliver_data(data)`
Reliable Data Transfer Protocol 1.0
Programming interfaces

- Application Layer: `data` `deliver_data()`
- Transport Layer:
  - `rdt_send()`
  - `make_pkt(pkt, data)`
  - `udt_send(pkt)`
  - `wait for call from above`
  - `wait for call from below`
  - `rdt_rcv(pkt)`
  - `extract(pkt, data)`
  - `deliver_data(data)`
- Network Layer:
  - `udt_send()`
  - `packet`
  - `Reliable Channel`

- This is the complete protocol under the assumption of a reliable network channel

Reliable Data Transfer Protocol 2.0
Reliable transfer over a channel with bit errors

- Now assume the underlying channel may “flip” random bits in a packet
- How to detect errors?
- How to recover from errors:
  - acknowledgements (ACKs) — the receiver explicitly tells the sender that a packet was received OK
  - negative acknowledgements (NAKs) — the receiver explicitly tells the sender that a packet had errors
  - Sender retransmits packet on receipt of NAK
- New mechanisms to deal with bit errors:
  - Error detection
  - Control messages (ACK, NAK) from a receiver to the sender
  - Retransmission
Reliable Data Transfer Protocol 2.0

Example 2: A corrupted packet arrives at the receiver

- Sender FSM
- Receiver FSM

- rdt_send(data)
- compute_checksum
- make_pkt(sndpkt, data, checksum)
- udt_send(sndpkt)

- rdt_rcv(rcvpkt)
- &
- isACK(rcvpkt)
- udt_send(sndpkt)

- rdt_rcv(rcvpkt)
- &
- notcorrupt(rcvpkt)
- extract(rcvpkt, data)
- deliver_data(data)
- udt_send(ACK)

- wait for call from above

- wait for ACK or NAK

- rdt_rcv(rcvpkt)
- &
- isNAK(rcvpkt)
- udt_send(NAK)

- wait for call from below

- rdt_rcv(rcvpkt)
- &
- corrupt(rcvpkt)
- udt_send(NAK)

- wait for call from above

- rdt_rcv(rcvpkt)
- &
- isNAK(rcvpkt)
- udt_send(NAK)

- rdt_rcv(rcvpkt)
- &
- notcorrupt(rcvpkt)
- udt_send(ACK)

- wait for call from below

Simple... but wrong!

- What happens if an ACK/NAK is corrupted?
  - Sender doesn’t know what happened at the receiver!

- What to do?
  - Sender ACKs/NAKs the receiver’s ACK/NAK?
  - Retransmit last data packet?
  - How to deal with duplicates?
Reliable Data Transfer Protocol 2.0
Simple… but wrong!

- Deal with corrupted ACKs/NAKs by retransmission of data packets
- Sender will add a sequence number to each packet to allow the receiver to detect duplicate packets
  - Receiver’s transport layer discards duplicate packets
- How much space to reserve in a header field for sequence numbers?

Reliable Data Transfer Protocol 2.1
Sender state machine to handle garbled ACKs/NAKs
Reliable Data Transfer Protocol 2.1
Receiver state machine to handle garbled ACKs/NAKs

<table>
<thead>
<tr>
<th>State</th>
<th>Actions</th>
</tr>
</thead>
</table>
| seq 0 | rdt_rcv(rcvptk)
& notcorrupt(rcvptk)
 & has_seq[rcvptk]
compute chksum
make_pkt(sndpkt,ACK,chksum)
udt_send(sndpkt) |
| seq 1 | rdt_rcv(rcvptk)
& notcorrupt(rcvptk)
 & has_seq[rcvptk]
extract(rcvptk,data)
deliver_data(data)
compute chksum
make_pkt(sndpkt,ACK,chksum)
udt_send(sndpkt) |

Discussion (Handling garbled ACKs/NAKs)

- **Sequence number added to header**
  - Two sequence numbers suffice
  - Must check if received ACK/NAK is corrupted
  - Number of states doubles
  - State encodes whether current packet has sequence number 0 or 1

- **Sender issues**
- **Receiver issues**

- **Must check if received packet is duplicate**
  - State encodes whether expected packet sequence number is 0 or 1

**Note:** Receiver cannot know if its last ACK/NAK received OK at sender.
Reliable Data Transfer Protocol 2.2

A NAK-free protocol

- Instead of NAKing, receiver sends ACK for last packet received OK
  - Receiver must include the sequence number of packet being ACKed in ACK
- Receipt of duplicate ACKs at sender is equivalent to a NAK
  - Sender retransmits current packet

Sender FSM

```
wait for rdt_send 0
wait for rdt_send 1

rdt_send(data)
compute checksum
make_pkt(sndpkt,0,data,chksum)
udt_send(sndpkt)

rdt_rcv(rcvpkt) && (corrupt(rcvpkt) || isACK(rcvpkt,1))
udt_send(sndpkt)

rdt_rcv(rcvpkt) && notcorrupt(rcvpkt)
&& isACK(rcv pkt,0)
udt_send(sndpkt)
```
**Reliable Data Transfer Protocol 3.0**

**Dealing with channels with errors and loss**

- Now assume the underlying channel can also **lose** packets
- New problem: How to detect loss?
  - Are checksums, ACKs, sequence numbers, retransmissions enough?
- Approach: sender waits “reasonable” amount of time and retransmits if no ACK received in this time
  - Requires the use of a countdown timer
- What if packet (or ACK) just delayed beyond its timer?
  - Retransmission will be **duplicate**…
  - But use of sequence numbers already handles this!

**Reliable Data Transfer Protocol 3.0**

**Sender state machine to handle lost/garbled packets**
**Receiver State Machine for RDT 2.2**

**What changes are needed to handle lost/garbled packets?**

- `rdt_rcv(rcvpkt) && notcorrupt(rcvpkt)`
- `&& has_seq0(rcvpkt)`
- `extract(rcvpkt.data)`
- `deliver_data(data)`
- `compute checksum`
- `make_pkt(sndpkt,ACK0,chksum)`
- `udt_send(sndpkt)`

- `rdt_rcv(rcvpkt) && notcorrupt(rcvpkt)`
- `&& has_seq1(rcvpkt)`
- `udt_send(sndpkt)`

- `wait for rdt_rcv seq 0`
- `wait for rdt_rcv seq 1`

**Reliable Data Transfer Protocol 3.0**

**Execution examples**

- **Protocol operation with no loss**
  - Sender: `send pkt0`
  - Receiver: `rcv pkt0`  `send pkt1`  `rcv pkt1`  `send ACK1`  `rcv pkt0`  `send ACK0`

- **Protocol operation with a lost packet**
  - Sender: `send pkt0`
  - Receiver: `rcv pkt0`  `send ACK0`
  - `pkt1` lost
  - `rcv pkt1`  `send ACK1`  `rcv pkt0`  `send ACK0`
Reliable Data Transfer Protocol 3.0

Execution examples

- Protocol operation with a lost ACK
- Protocol operation with a poor timeout value

Reliable Data Transfer Protocol 3.0

Sender state machine to handle lost/garbled packets

- rdt_send(data)
- rdt_rcv(rcvpkt)
- wait for call0 from above
- timeout
- start_timer

- compute checksum
- make_pkt(sndpkt,0,data,checksum)
- rdt_send(sndpkt)

- rdt_rcv(rcvpkt) && isACK(rcvpkt,0)
- stop_timer

- compute checksum
- make_pkt(sndpkt,1,data,checksum)
- rdt_send(sndpkt)
- start_timer

- rdt_rcv(rcvpkt) && isACK(rcvpkt,1)
- wait for ACK 1
- start_timer