Last week on Communication Networks

We explored the concepts behind routing

How do you guide IP packets from a source to destination?

Upon packet reception, routers locally look up their forwarding table to know where to send it next

Here, the packet should be directed to IF#4
Forwarding is repeated at each router, until the destination is reached.

Routing is the control-plane process that computes and populates the forwarding tables

Forwarding vs Routing
summary

<table>
<thead>
<tr>
<th>forwarding</th>
<th>routing</th>
</tr>
</thead>
<tbody>
<tr>
<td>goal</td>
<td>computing the paths packets will follow</td>
</tr>
<tr>
<td>scope</td>
<td>network-wide</td>
</tr>
<tr>
<td>impl.</td>
<td>software</td>
</tr>
<tr>
<td>timescale</td>
<td>10s of ms hopefully</td>
</tr>
</tbody>
</table>

The goal of routing is to compute valid global forwarding state

- definition a global forwarding state is valid if it always delivers packets to the correct destination

sufficient and necessary condition

- observation 1 Verifying that a forwarding state is valid is easy
- observation 2 There are 3 ways to compute valid forwarding state

There are three ways to compute valid routing state

<table>
<thead>
<tr>
<th>Intuition</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>Use tree-like topologies</td>
</tr>
<tr>
<td>#2</td>
<td>Rely on a global network view</td>
</tr>
<tr>
<td>#3</td>
<td>Rely on distributed computation</td>
</tr>
</tbody>
</table>

This week on Communication Networks
**Reliable Transport**

1. Correctness condition: if-and-only if again
2. Design space: timeliness vs efficiency vs ...
3. Examples: Go-Back-N & Selective Repeat

In the Internet, reliability is ensured by the end hosts, not by the network.

Internet layer puts reliability in L4, just above the Network layer.

Internet layer puts reliability in L4, just above the Network layer.

Recall that the Network provides a best-effort service, with quite poor guarantees.

IP packets can get lost or delayed.

IP packets can get corrupted.
IP packets can get reordered

Alice  Internet  Bob

packet 1
packet 2
packet 3

packet 1
packet 2
packet 3

IP packets can get duplicated

Alice  Internet  Bob

packet 1
packet 2
packet 3

packet 1
packet 2
packet 3
packet 1
packet 1
packet 3

The four goals of reliable transfer

correctness  ensure data is delivered, in order, and untouched
timeliness  minimize time until data is transferred
efficiency  optimal use of bandwidth
fairness  play well with concurrent communications

goals

Routing had a clean sufficient and necessary correctness condition

Theorem  a global forwarding state is valid if and only if

- there are no dead ends
  no outgoing port defined in the table
- there are no loops
  packets going around the same set of nodes

We need the same kind of “if and only if” condition for a “correct” reliable transport design

A reliable transport design is correct if...

attempt #1  packets are delivered to the receiver

Wrong  Consider that the network is partitioned

We cannot say a transport design is incorrect if it doesn’t work in a partitioned network...

Wrong  If the network is only available one instant in time, only an oracle would know when to send

A reliable transport design is correct if...

attempt #2  packets are delivered to receiver if and only if it was possible to deliver them

Wrong  We cannot say a transport design is incorrect if it doesn’t know the unknowable
A reliable transport design is correct if...

**attempt #3**
It resends a packet if and only if
the previous packet was lost or corrupted

Wrong
Consider two cases
- packet made it to the receiver and
  all packets from receiver were dropped
- packet is dropped on the way and
  all packets from receiver were dropped

A reliable transport design is correct if...

**attempt #3**
It resends a packet if and only if
the previous packet was lost or corrupted

Wrong
In both case, the sender has no feedback at all

Wrong
but better as it refers to what the design does (which it can control),
not whether it always succeeds (which it can't)
There is a clear tradeoff between timeliness and efficiency in the selection of the timeout value.

Timeliness argues for small timers, efficiency for large ones.

An obvious solution to improve timeliness is to send multiple packets at the same time.

Sending multiple packets improves timeliness, but it can also overwhelm the receiver.

To solve this issue, we need a mechanism for flow control.
Using a sliding window is one way to do that:

- Sender keeps a list of the sequence # it can send known as the sending window.
- Receiver also keeps a list of the acceptable sequence # known as the receiving window.
- Sender and receiver negotiate the window size.

Sender and receiver negotiate the window size such that:

\[
\text{sending window} \leq \text{receiving window}
\]

Example with a window composed of 4 packets:

<table>
<thead>
<tr>
<th>unACK'ed packets</th>
<th>forbidden packets</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4 5 6 7 8 9 10 11 ...</td>
<td></td>
</tr>
</tbody>
</table>

Example with a window composed of 4 packets:

<table>
<thead>
<tr>
<th>ACKed packets</th>
<th>available packets</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4 5 6 7 8 9 10 11 ...</td>
<td></td>
</tr>
</tbody>
</table>

Window after sender receives ACK 4:

<table>
<thead>
<tr>
<th>ACKed packets</th>
<th>available packets</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4 5 6 7 8 9 10 11 ...</td>
<td></td>
</tr>
</tbody>
</table>

Timeliness of the window protocol depends on the size of the sending window.

Assuming infinite buffers, how big should the window be to maximize timeliness?

Alice

| 100 Mbps, 5 ms |

What should be the value of W?

(in bytes)

Bob

Timeliness matters, but what about efficiency?

The efficiency of our protocol essentially depends on two factors:

- How much information does the sender get?
- How does the sender detect and react to losses?
The efficiency of our protocol essentially depends on two factors

![receiver feedback]

How much information does the sender get?

ACKing individual packets provides detailed feedback, but triggers unnecessary retransmission upon losses

<table>
<thead>
<tr>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>know fate of each packet</td>
<td>loss of an ACK packet requires a retransmission</td>
</tr>
<tr>
<td>simple window algorithm W single-packet algorithms</td>
<td>causes unnecessary retransmission</td>
</tr>
<tr>
<td>not sensitive to reordering</td>
<td></td>
</tr>
</tbody>
</table>

Cumulative ACKs enables to recover from lost ACKs, but provides coarse-grained information to the sender

<table>
<thead>
<tr>
<th>Approach</th>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACK the highest sequence number for which all the previous packets have been received</td>
<td>recover from lost ACKs</td>
<td>confused by reordering incomplete information about which packets have arrived causes unnecessary retransmission</td>
</tr>
</tbody>
</table>

Full Information Feedback prevents unnecessary retransmission, but can induce a sizable overhead

<table>
<thead>
<tr>
<th>Approach</th>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>List all packets that have been received highest cumulative ACK, plus any additional packets</td>
<td>complete information resilient form of individual ACKs</td>
<td>overhead (hence lowering efficiency) e.g., when large gaps between received packets</td>
</tr>
</tbody>
</table>

We see that Internet design is all about balancing tradeoffs (again)

The efficiency of our protocol essentially depends on two factors

![receiver feedback]

How does the sender detect and react to losses?

As of now, we detect loss by using timers. That’s only one way though

Losses can also be detected by relying on ACKs
With individual ACKs, missing packets (gaps) are implicit

| ACK stream | 1 | 2 | 3 | 4 | 5 | 6 | 7 | ...
|-------------|---|---|---|---|---|---|---|---
| sender can infer that 5 is missing and resend after k subsequent packets |

With full information, missing packets (gaps) are explicit

<table>
<thead>
<tr>
<th>ACK stream</th>
<th>up to 1</th>
<th>up to 2</th>
<th>up to 3</th>
<th>up to 4</th>
<th>sender learns that 5 is missing and retransmits after k packets</th>
</tr>
</thead>
<tbody>
<tr>
<td>up to 4, plus 6</td>
<td>up to 4, plus 6—7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

With cumulative ACKs, missing packets are harder to know

<table>
<thead>
<tr>
<th>ACK stream</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>4</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 sent when 6 arrives</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 sent when 7 arrives</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Duplicated ACKs are a sign of isolated losses. Dealing with them is trickier though.

<table>
<thead>
<tr>
<th>situation</th>
<th>Lack of ACK progress means that 5 hasn’t made it</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stream of ACKs means that (some) packets are delivered</td>
<td></td>
</tr>
<tr>
<td>Sender could trigger resend upon receiving k duplicates ACKs</td>
<td></td>
</tr>
<tr>
<td>but what do you resend?</td>
<td></td>
</tr>
<tr>
<td>only 5 or 5 and everything after?</td>
<td></td>
</tr>
</tbody>
</table>

What about fairness?

Design a correct, timely, efficient and fair transport mechanism knowing that packets can get lost, corrupted, reordered, delayed, duplicated.

When n entities are using our transport mechanism, we want a fair allocation of the available bandwidth

An equal allocation is certainly “fair”, but what about the efficiency of the network?

Consider this simple network in which three hosts are sharing two links

A 1Gbps B 1Gbps C

Flow 1

Flow 2

Flow 3

What is a fair allocation for the 3 flows?

A 1Gbps B 1Gbps C

Flow 1 500 Mbps

Flow 2 500 Mbps

Flow 3 500 Mbps

Total traffic is 1.5 Gbps
Fairness and efficiency don’t always play along, here an unfair allocation ends up more efficient.

Total traffic is 2 Gbps!

What is fair anyway?

With equal-per-flow, A ends up with 1 Gbps because it sends 2 flows, while B ends up with 500 Mbps

Is it fair?

Seeking an exact notion of fairness is not productive. What matters is to avoid starvation.

equal per flow is good enough for this

Intuitively, we want to give users with “small” demands what they want, and evenly distribute the rest.

Max-min fair allocation can easily be computed

Max-min fair allocation is such that

the lowest demand is maximized

after the lowest demand has been satisfied, the second lowest demand is maximized

after the second lowest demand has been satisfied, the third lowest demand is maximized

and so on…

step 1: Start with all flows at rate 0

step 2: Increase the flows until there is a new bottleneck in the network

step 3: Hold the fixed rate of the flows that are bottlenecked

step 4: Go to step 2 for the remaining flows

Done!
Let’s try on this network

![Network Diagram]

What’s the max-min fair allocation?

Max-min fair allocation can be approximated by slowly increasing $W$ until a loss is detected

- **Intuition**: Progressively increase the sending window size.
- **Progressively increase the sending window size**
- **Whenever a loss is detected**, decrease the window size
- **Repeat**

Dealing with **corruption** is easy:
- Rely on a checksum, treat corrupted packets as lost

Design a correct, timely, efficient and fair transport mechanism knowing that lost packets can get corrupted, reordered, delayed, duplicated

The effect of **reordering** depends on the type of ACKing mechanism used

- **individual ACKs**: no problem
- **full feedback**: no problem
- **cumm. ACKs**: create duplicate ACKs
  - why is it a problem?

Long delays can create useless timeouts, for all designs

Packets **duplicates** can lead to duplicate ACKs whose effects will depend on the ACKing mechanism used

- **individual ACKs**: no problem
- **full feedback**: no problem
- **cumm. ACKs**: problematic

Design a correct, timely, efficient and fair transport mechanism knowing that packets can get lost, corrupted, reordered, delayed, duplicated
Here is one correct, timely, efficient and fair transport mechanism:

**ACKing**
- full information ACK

**retransmission**
- after timeout
- after \(k\) subsequent ACKs

**window management**
- additive increase upon successful delivery
- multiple decrease when timeouts

We’ll come back to this when we see TCP.

Go-Back-N (GBN) is a simple sliding window protocol using cumulative ACKs:

**principle**
- receiver should be as simple as possible

**receiver**
- delivers packets in-order to the upper layer for each received segment, ACK the last in-order packet delivered (cumulative)

**sender**
- use a single timer to detect loss, reset at each ACK upon timeout, resend all W packets starting with the lost one

Finite State Machine for the receiver:

- see Book 3.4.3

Finite State Machine for the sender:

- see Book 3.4.3

Selective Repeat (SR) avoid unnecessary retransmissions by using per-packet ACKs:

**principle**
- avoids unnecessary retransmissions

**receiver**
- acknowledge each packet, in-order or not buffer out-of-order packets

**sender**
- use per-packet timer to detect loss upon loss, only resend the lost packet

Reliable Transport:

**Correctness condition**
- if and only if again

**Design space**
- timeliness vs efficiency vs ...

**Examples**
- Go-Back-N & Selective Repeat