Communication Networks
Prof. Laurent Vanbever

Last week on Communication Networks

How it works
Protocol
1
Policies
“Follow the money”
2
Problems
Security, performance, …
3

Border Gateway Protocol policies and more

This week on Communication Networks

How do you ensure reliable transport on top of best-effort delivery?

Let’s look back at an example of BGP hijack
Without RPKI, a more-specific attack by AS34109 successfully manages to attract the traffic.

The hijacked announcement is now INVALID.

Let's assume now that AS 29997 registers (204.16.254.0/24–32, 29997) as a new ROA.

But what if AS34109 announce AS29997 as the origin?

Here greenhost.nl receives 2 valid RPKI routes: one via NTT and another one via 34109.

As the route via 34109 has a shorter path, it is preferred... the attack works again!

We see that RPKI does not protect against all attacks.

Communication Networks
Part 2: Concepts

How do you ensure reliable transport on top of best-effort delivery?

Source: Sharon Goldberg, "The Transition to BGP Security. Is the Juice Worth the Squeeze?"
In the Internet, reliability is ensured by the end hosts, not by the network.

The Internet puts reliability in L4, just above the network layer.

### Goals
- Keep the network simple, dumb
- Make it relatively "easy" to build and operate a network
- Keep applications as network "unaware" as possible
- A developer should focus on its app, not on the network

### Design
- Implement reliability in-between, in the networking stack
- Relieve the burden from both the app and the network

Application
Transport
Network
Link
Physical

**Layer**
- L4: Transport
- L3: Network
- L2: Link
- L1: Physical

The Internet puts reliability in L4, just above the network layer.

Recall that the network layer provides a best-effort service only.

Application
Transport
Network
Link
Physical

**Layer**
- L4: Transport
- L3: Network
- L2: Link
- L1: Physical

Let's consider a simple communication between two end-points, Alice and Bob.

IP packets can get lost or delayed.

Alice  →  Internet  →  Bob

IP packets can get corrupted.

payload: 101
 payload: 010
 payload: 42

Alice  →  Internet  →  Bob

IP packets can get reordered.

Alice  →  Internet  →  Bob
IP packets can get duplicated.

Reliable Transport

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

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

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...
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
- If the network is only available one instant in time, only an oracle would know when to send.
- We cannot say a transport design is incorrect if it doesn’t know the unknowable.

**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.

Wrong
- In both cases, the sender has no feedback at all. Does it resend or not?

Wrong
- But better as it refers to what the design does (which it can control), not whether it always succeeds (which it can’t).

**Attempt #4**
- A packet is always resent if the previous packet was lost or corrupted.
- A packet may be resent at other times.

Correct!

Now, that we have a correctness condition, how do we achieve it and with what tradeoffs?

A transport mechanism is correct if and only if it resends all dropped or corrupted packets.

Sufficient
- “If” algorithm will always keep trying to deliver undelivered packets.

Necessary
- “only if” if it ever let a packet go undelivered without resending it, it isn’t reliable.

Note
- It is ok to give up after a while but must announce it to the application.

Reliable Transport

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

Let’s focus on these aspects first!

- Design a correct, timely, efficient and fair transport mechanism knowing that packets can get lost, corrupted, reordered, delayed, duplicated.
Timeliness argues for small timers, efficiency for large ones.

There is a clear tradeoff between timeliness and efficiency in the selection of the timeout value.

Even with short timers, the timeliness of our protocol is extremely poor: one packet per Round-Trip Time (RTT).

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 sending window <= 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</td>
<td>8 9 10 11 ...</td>
</tr>
</tbody>
</table>

ACKed packets available packets forbidden packets

Window after sender receives ACK 4

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 (one-way)

Bob

What should be the value of W? (in bits)

Timeliness matters, but what about efficiency?

The efficiency of our protocol essentially depends on two factors:

- receiver feedback
- behavior upon losses

How much information does the sender get?

How does the sender detect and react to losses?
**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</td>
<td>Causes unnecessary retransmission</td>
</tr>
<tr>
<td>W single-packet algorithms</td>
<td>Not sensitive to reordering</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></th>
</tr>
</thead>
<tbody>
<tr>
<td>Ack the highest sequence number for which all the previous packets have been received</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recover from lost ACKs</td>
<td>Confused by reordering</td>
</tr>
<tr>
<td>Incomplete information about which packets have arrived</td>
<td>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></th>
</tr>
</thead>
<tbody>
<tr>
<td>List all packets that have been received highest cumulative ACK, plus any additional packets</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complete information</td>
<td>Overhead (hence lowering efficiency)</td>
</tr>
<tr>
<td>Resilient form of individual ACKs</td>
<td>E.g., when large gaps between received packets</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Approach</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>List all packets that have been received highest cumulative ACK, plus any additional packets</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complete information</td>
<td>Overhead (hence lowering efficiency)</td>
</tr>
<tr>
<td>Resilient form of individual ACKs</td>
<td>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
- Behavior upon losses

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**

Assume packet 5 is lost but no other

<table>
<thead>
<tr>
<th>ACK stream</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>sender can infer that 5 is missing and resend 5 after k subsequent packets</td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

**With individual ACKs, missing packets (gaps) are implicit**
With full information, missing packets (gaps) are explicit

Assume packet 5 is lost
but no other

ACK stream
up to 1
up to 2
up to 3
up to 4
up to 4, plus 6
up to 4, plus 6 to 7
sender learns that 5 is missing
retransmits after 8 packets

With cumulative ACKs, missing packets are harder to know

Assume packet 5 is lost
but no other

ACK stream
1
2
3
4
4 sent when 6 arrives
4 sent when 7 arrives

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

Lack of ACK progress means that 5 hasn’t made it
Stream of ACKs means that (some) packets are delivered

Sender could trigger resend:
upon receiving 8 duplicates ACKs
but what do you resend?
only 5 or 5 and everything after?

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

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

What is a fair allocation for the 3 flows?

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

Fairness and efficiency don’t always play along, here an unfair allocation ends up more efficient

Total traffic is 1.5 Gbps

Total traffic is 2 Gbps!
What is fair anyway?

Equal-per-flow isn’t really fair as (A,C) crosses two links: it uses more resources.

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

Simply dividing the available bandwidth doesn’t work in practice since flows can see different bottleneck

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

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...

Max-min fair allocation can easily be computed

Let’s try on this network

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.

Whenever a loss is detected, decrease the window size.

Repeat.

---

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

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

**The effect of reordering** depends on the type of ACKing mechanism used:
- Individual ACKs: no problem
- Full feedback: no problem
- Cumulative ACKs: create duplicate ACKs

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
- Cumulative ACKs: problematic

---

Here is one correct, timely, efficient and fair transport mechanism:

- **ACKing**: full information ACK
- **Retransmission**: after timeout
- **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

**design space**
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 new ACK
upon timeout, resend all W packets starting with the lost one

Let's see how it works in practice visually

http://www.ccs-labs.org/teaching/rn/animations/gbn_sr/

Selective Repeat (SR) avoid unnecessary retransmissions by using per-packet ACKs see Book 3.4.3

**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