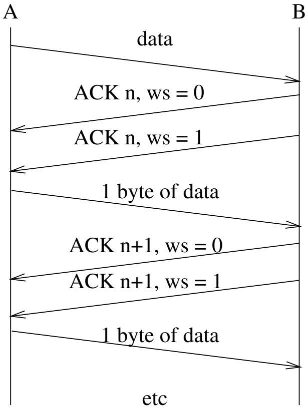
TCP Strategies

Silly Window Syndrome

Another problem with tinygrams is manifested as *silly window syndrome*

TCP Strategies

Silly Window Syndrome

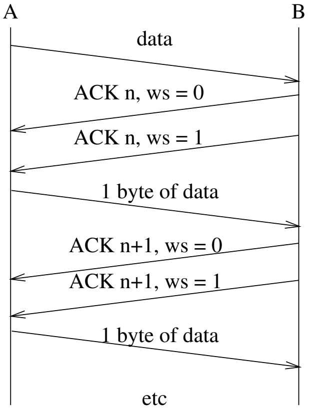


Silly Window Syndrome

A is sending data to B, but B's buffer is nearly full and B is reading only one byte at a time;

TCP Strategies

Silly Window Syndrome

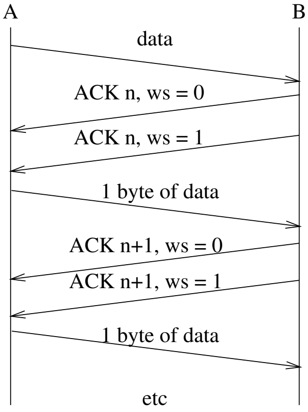


Silly Window Syndrome

B's buffer fills, and B ACKs with a window of 0;

TCP Strategies

Silly Window Syndrome

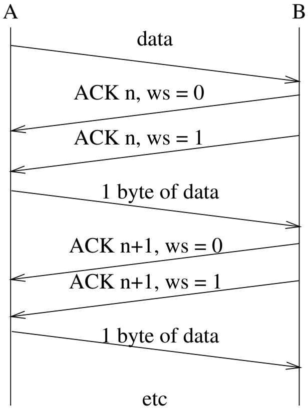


Silly Window Syndrome

A holds off sending more data;

TCP Strategies

Silly Window Syndrome

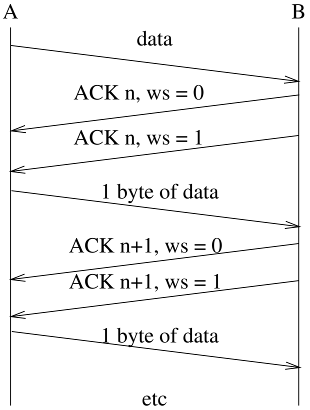


Silly Window Syndrome

B reads a byte;

TCP Strategies

Silly Window Syndrome

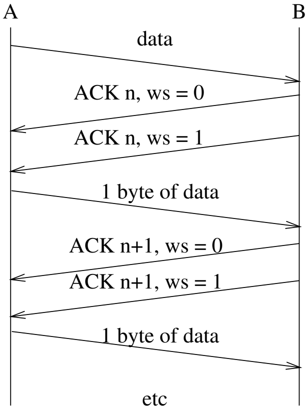


Silly Window Syndrome

B sends a window update segment, size 1;

TCP Strategies

Silly Window Syndrome

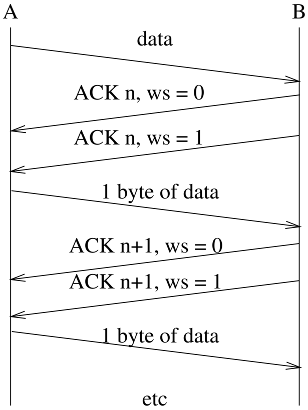


Silly Window Syndrome

A get this and sends as much data as possible, i.e., 1 byte;

TCP Strategies

Silly Window Syndrome

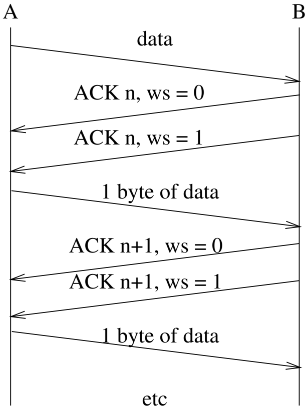


Silly Window Syndrome

B ACKs with window 0;

TCP Strategies

Silly Window Syndrome

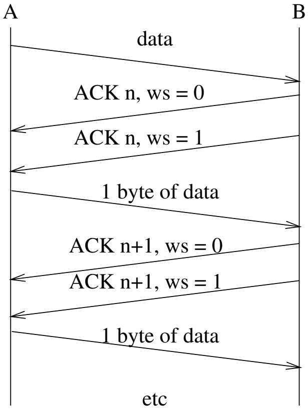


Silly Window Syndrome

B reads a byte;

TCP Strategies

Silly Window Syndrome

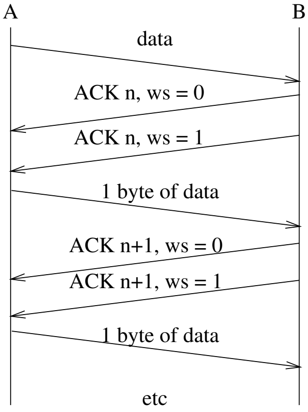


Silly Window Syndrome

B sends an update, size 1;

TCP Strategies

Silly Window Syndrome

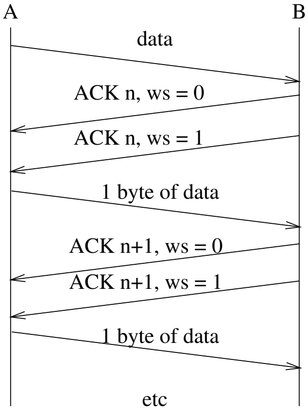


Silly Window Syndrome

A sends 1 byte;

TCP Strategies

Silly Window Syndrome



Silly Window Syndrome

And so on

TCP Strategies

Silly Window Syndrome

We are back to the two segment per byte high overhead: this is silly window syndrome

TCP Strategies

Silly Window Syndrome

We are back to the two segment per byte high overhead: this is silly window syndrome

Better is for B not to send an update of 1, but wait until there is more space

TCP Strategies

Silly Window Syndrome

We are back to the two segment per byte high overhead: this is silly window syndrome

Better is for B not to send an update of 1, but wait until there is more space

Clarke's algorithm to avoid SWS is in the server

> *never send an update for a window of 1; only advertise a new window when either (a) there is enough space for a full segment, or (b) the buffer is half empty*

TCP Strategies

Congestion

Nagle (in the client) and SWS (in the server) fit together naturally

TCP Strategies

Congestion

Nagle (in the client) and SWS (in the server) fit together naturally

Note that TCP code doesn’t have to implement Nagle or SWS or delayed ACKs or any of these strategies: it’s just a good idea if it does!

TCP Strategies

Congestion

Nagle and SWS are good for when there is a small amount of data being transmitted

TCP Strategies

Congestion

Nagle and SWS are good for when there is a small amount of data being transmitted

We need to look at the case of sending large amounts of data

TCP Strategies

Congestion

Nagle and SWS are good for when there is a small amount of data being transmitted

We need to look at the case of sending large amounts of data

We want the data to get to the destination as fast as possible, but we now have to consider not just the ability of the destination to cope, but also the capacity of the network itself

TCP Strategies

Congestion

Congestion happens when more data is being sent than the *network* can handle: routers will drop packets if there is not enough onward bandwidth to cope

TCP Strategies

Congestion

Congestion happens when more data is being sent than the *network* can handle: routers will drop packets if there is not enough onward bandwidth to cope

There are several strategies in TCP to help deal with and avoid congestion

TCP Strategies

Congestion

Congestion happens when more data is being sent than the *network* can handle: routers will drop packets if there is not enough onward bandwidth to cope

There are several strategies in TCP to help deal with and avoid congestion

The first issue is how to spot congestion, given that it might be happening in a part of the network many hops away from both source and destination

TCP Strategies

Congestion

We watch for segment loss

TCP Strategies

Congestion

We watch for segment loss

Segments can be lost though errors in transmission or being dropped at a congested router (or at the destination)

TCP Strategies

Congestion

We watch for segment loss

Segments can be lost though errors in transmission or being dropped at a congested router (or at the destination)

Poor transmission is less usual these days, so we can assume loss is due to congestion (which is common these days)

TCP Strategies

Congestion

We watch for segment loss

Segments can be lost though errors in transmission or being dropped at a congested router (or at the destination)

Poor transmission is less usual these days, so we can assume loss is due to congestion (which is common these days)

Thus TCP treats missing or duplicate ACKs as a sign of congestion

TCP Strategies

Congestion

We watch for segment loss

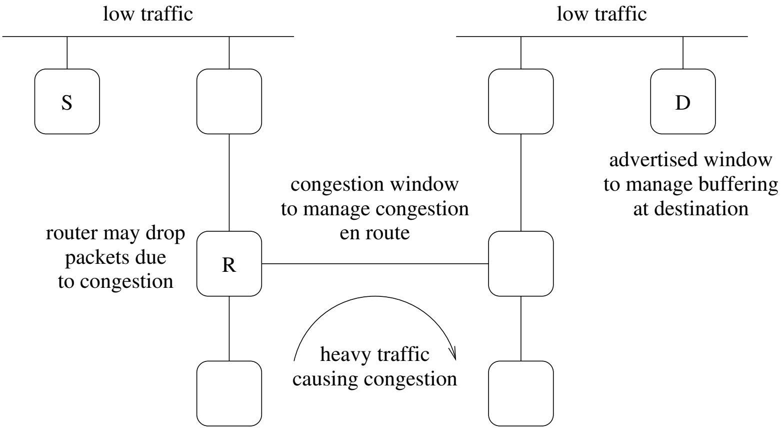
Segments can be lost though errors in transmission or being dropped at a congested router (or at the destination)

Poor transmission is less usual these days, so we can assume loss is due to congestion (which is common these days)

Thus TCP treats missing or duplicate ACKs as a sign of congestion

Exercise A missing ACK is understandable as a sign of congestion: reflect briefly on why *duplicate* ACKs can be caused by congestion

TCP Strategies



Congestion somewhere on the path

Congestion can happen in a router due to lack of capacity in an onward link; a router will drop a packet if it can't cope

TCP Strategies

Congestion

Just as the advertised window deals with “congestion in the destination” (it’s not really congestion), we have the *congestion window* for congestion in the network

TCP Strategies

Congestion

Just as the advertised window deals with “congestion in the destination” (it’s not really congestion), we have the *congestion window* for congestion in the network

So how do we determine the congestion window? It’s not a thing the source or destination can know directly

TCP Strategies

Congestion

Just as the advertised window deals with “congestion in the destination” (it’s not really congestion), we have the *congestion window* for congestion in the network

So how do we determine the congestion window? It’s not a thing the source or destination can know directly

We do this by sending segments and watching what ACKs we get

TCP Strategies

Congestion

If we have a lot of data to send we do not want to wait for each ACK before sending the next segment

TCP Strategies

Congestion

If we have a lot of data to send we do not want to wait for each ACK before sending the next segment

Better is to send several segments and then wait to see from the ACKs which were safely received

TCP Strategies

Congestion

But sending too many segments at once is bad when the network is congested: our segments will be dropped. We'll just be making things worse for everyone, including ourselves

TCP Strategies

Congestion

But sending too many segments at once is bad when the network is congested: our segments will be dropped. We'll just be making things worse for everyone, including ourselves

So, if we have an estimate of the capacity of the network (the congestion window), we will be sending many segments at once, but not too many

TCP Strategies

Congestion

But sending too many segments at once is bad when the network is congested: our segments will be dropped. We'll just be making things worse for everyone, including ourselves

So, if we have an estimate of the capacity of the network (the congestion window), we will be sending many segments at once, but not too many

If we get it right, we will have a continual stream of segments going out and ACKs coming back

TCP Strategies

Slow Start & Congestion Avoidance

We estimate the network congestion by watching the number of ACKs coming back

TCP Strategies

Slow Start & Congestion Avoidance

We estimate the network congestion by watching the number of ACKs coming back

This estimate controls the congestion window

TCP Strategies

Slow Start & Congestion Avoidance

We estimate the network congestion by watching the number of ACKs coming back

This estimate controls the congestion window

This is an another constraint on sending additional to the advertised window: it’s a bad idea to send more data than indicated by the either window

TCP Strategies

Slow Start & Congestion Avoidance

We describe a basic flow control strategy (RFC2001/RFC2581) that estimates the congestion window; many modifications exist (TCP Tahoe, TCP Reno, TCP Vegas, ...)

TCP Strategies

Slow Start & Congestion Avoidance

We describe a basic flow control strategy (RFC2001/RFC2581) that estimates the congestion window; many modifications exist (TCP Tahoe, TCP Reno, TCP Vegas, ...)

The *congestion window* (`cwnd`) is initialised to the maximum segment (MSS) size of the destination

TCP Strategies

Slow Start & Congestion Avoidance

We describe a basic flow control strategy (RFC2001/RFC2581) that estimates the congestion window; many modifications exist (TCP Tahoe, TCP Reno, TCP Vegas, ...)

The *congestion window* (`cwnd`) is initialised to the maximum segment (MSS) size of the destination

A variable, `ssthresh`, the *threshold*, is initialised to 64KB (say)

TCP Strategies

Slow Start & Congestion Avoidance

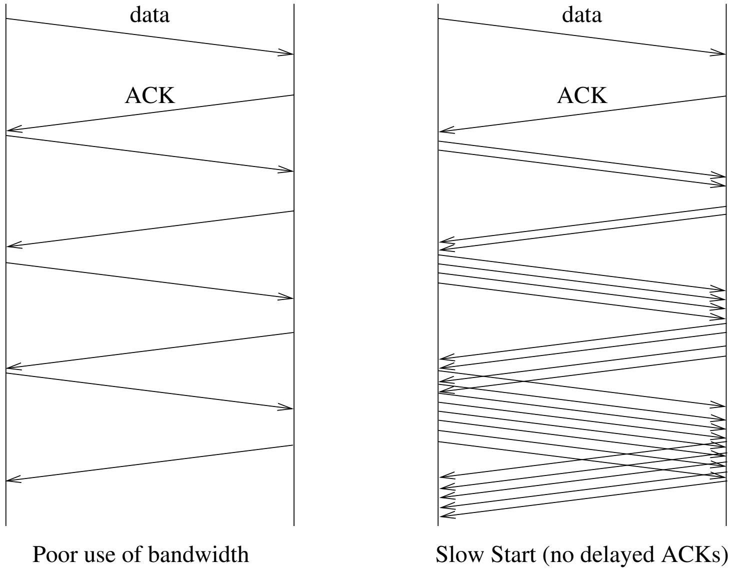
We describe a basic flow control strategy (RFC2001/RFC2581) that estimates the congestion window; many modifications exist (TCP Tahoe, TCP Reno, TCP Vegas, . . .)

The *congestion window* (`cwnd`) is initialised to the maximum segment (MSS) size of the destination

A variable, `ssthresh`, the *threshold*, is initialised to 64KB (say)

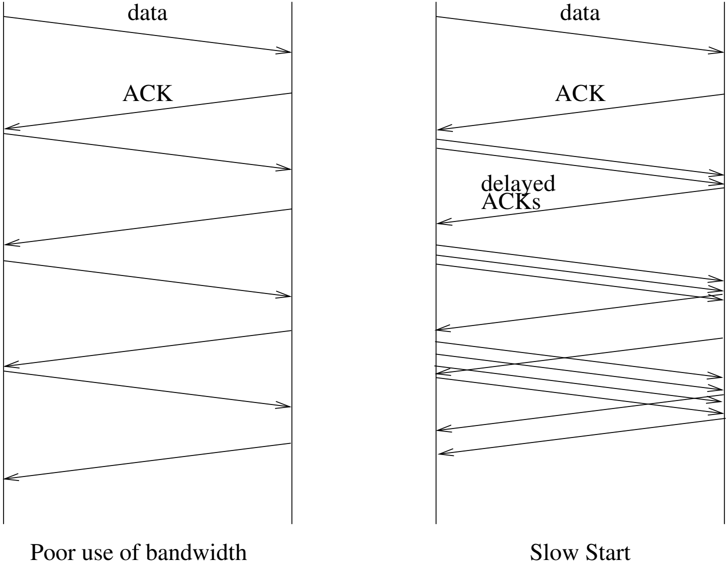
Every time a timely ACK is received, the congestion window is increased by one segment

TCP Strategies



Slow Start with no delayed ACKs

TCP Strategies



Slow Start with delayed ACKs

TCP Strategies

Slow Start & Congestion Avoidance

So initially we send one segment

TCP Strategies

Slow Start & Congestion Avoidance

So initially we send one segment

Then two at a time

TCP Strategies

Slow Start & Congestion Avoidance

So initially we send one segment

Then two at a time

Then four…

TCP Strategies

Slow Start & Congestion Avoidance

So initially we send one segment

Then two at a time

Then four…

This is called *slow start*

TCP Strategies

Slow Start & Congestion Avoidance

It is actually a near-exponential increase in the congestion window over time

TCP Strategies

Slow Start & Congestion Avoidance

It is actually a near-exponential increase in the congestion window over time

It is “slow” in comparison with an earlier version of TCP that started by blasting out segments as fast as possible before the performance of the network was known

TCP Strategies

Slow Start & Congestion Avoidance

It is actually a near-exponential increase in the congestion window over time

It is “slow” in comparison with an earlier version of TCP that started by blasting out segments as fast as possible before the performance of the network was known

In slow start, the increase continues until we reach the current threshold `ssthresh` or returning ACKs are duplicated or timed out

TCP Strategies

Slow Start & Congestion Avoidance

Of course, the rate is also limited by the advertised window of the destination: we can only send the minimum of the current congestion window and the advertised window

TCP Strategies

Slow Start & Congestion Avoidance

Of course, the rate is also limited by the advertised window of the destination: we can only send the minimum of the current congestion window and the advertised window

Note that the congestion window is a limit set by the sender, while the advertised window is a limit set by the receiver

TCP Strategies

Slow Start & Congestion Avoidance

If we reach `ssthresh` without a problem, we change to the *congestion avoidance* phase

TCP Strategies

Slow Start & Congestion Avoidance

If we reach `ssthresh` without a problem, we change to the *congestion avoidance* phase

Now we increase the congestion window `cwnd` by one segment for each round trip time (RTT)

TCP Strategies

Slow Start & Congestion Avoidance

If we reach `ssthresh` without a problem, we change to the *congestion avoidance* phase

Now we increase the congestion window `cwnd` by one segment for each round trip time (RTT)

So one per burst of segments

TCP Strategies

Slow Start & Congestion Avoidance

If we reach `ssthresh` without a problem, we change to the *congestion avoidance* phase

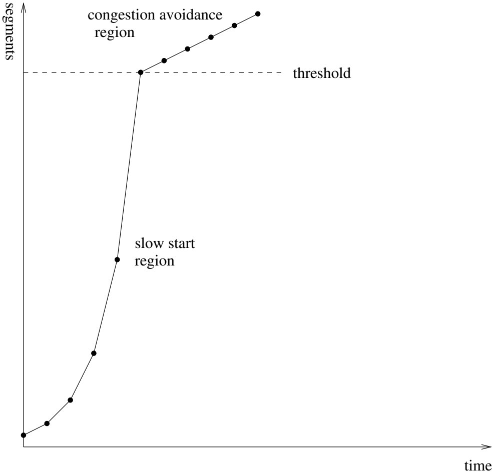
Now we increase the congestion window `cwnd` by one segment for each round trip time (RTT)

So one per burst of segments

This is now a linear increase over time

TCP Strategies

Slow Start & Congestion Avoidance



Slow start and congestion avoidance regions

TCP Strategies

Slow Start & Congestion Avoidance

Eventually the network's limit will be reached and a congested router somewhere will start dropping segments

TCP Strategies

Slow Start & Congestion Avoidance

Eventually the network's limit will be reached and a congested router somewhere will start dropping segments

The sender will see this when either (a) it gets some duplicate ACKs, or (b) there is a timeout waiting for ACKs

TCP Strategies

Slow Start & Congestion Avoidance

Eventually the network's limit will be reached and a congested router somewhere will start dropping segments

The sender will see this when either (a) it gets some duplicate ACKs, or (b) there is a timeout waiting for ACKs

Note we might be in either of the slow start or the congestion avoidance phases when congestion occurs: particularly if `ssthresh` was initially set very large, as its often done these days

TCP Strategies

Slow Start & Congestion Avoidance

When congestion is detected

TCP Strategies

Slow Start & Congestion Avoidance

When congestion is detected

- the threshold `ssthresh` is set to half the current transmit size. This is the smaller of the current congestion window and the advertised window. Also, this is rounded up to a minimum of two segments

TCP Strategies

Slow Start & Congestion Avoidance

When congestion is detected

- the threshold `ssthresh` is set to half the current transmit size. This is the smaller of the current congestion window and the advertised window. Also, this is rounded up to a minimum of two segments
- if it was a timeout, the congestion window `cwnd` is set back to one segment, and go back into slow start

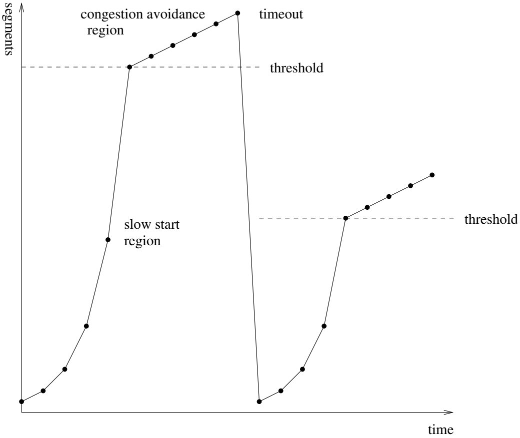
TCP Strategies

Slow Start & Congestion Avoidance

When congestion is detected

- the threshold `ssthresh` is set to half the current transmit size. This is the smaller of the current congestion window and the advertised window. Also, this is rounded up to a minimum of two segments
- if it was a timeout, the congestion window `cwnd` is set back to one segment, and go back into slow start
- when ACKs start coming through, we resume increasing the congestion window again, according to whether we were in slow start or congestion avoidance (i.e., whether `cwnd` is less than `ssthresh` or not)

TCP Strategies



Converging on the optimum rate

The sender eventually converges on a rate that is neither too fast, nor too slow

TCP Strategies
Slow Start & Congestion Avoidance

And it is dynamic

TCP Strategies

Slow Start & Congestion Avoidance

And it is dynamic

If conditions on the network change, it will soon adapt to the new rate, be it faster or slower

TCP Strategies

Slow Start & Congestion Avoidance

And it is dynamic

If conditions on the network change, it will soon adapt to the new rate, be it faster or slower

If there is no congestion on the network, the rate increases until it reaches the advertised window: the limiting factor is then the destination, not the network

TCP Strategies

Slow Start & Congestion Avoidance

And it is dynamic

If conditions on the network change, it will soon adapt to the new rate, be it faster or slower

If there is no congestion on the network, the rate increases until it reaches the advertised window: the limiting factor is then the destination, not the network

This strategy is very effective: get the flow up quickly, but don't overshoot network capacity. Also, back off quickly and try again when a loss happens

TCP Strategies

Fast Retransmit

As previously mentioned, when an out-of-order segment is received the TCP protocol calls for an immediate (possibly duplicate) ACK: it must not be delayed

TCP Strategies

Fast Retransmit

As previously mentioned, when an out-of-order segment is received the TCP protocol calls for an immediate (possibly duplicate) ACK: it must not be delayed

Thus the sender will start seeing duplicate ACKs

TCP Strategies

Fast Retransmit

As previously mentioned, when an out-of-order segment is received the TCP protocol calls for an immediate (possibly duplicate) ACK: it must not be delayed

Thus the sender will start seeing duplicate ACKs

This is to inform the sender as soon as possible that something is wrong

TCP Strategies

Fast Retransmit

As previously mentioned, when an out-of-order segment is received the TCP protocol calls for an immediate (possibly duplicate) ACK: it must not be delayed

Thus the sender will start seeing duplicate ACKs

This is to inform the sender as soon as possible that something is wrong

Jacobson's Fast Retransmit strategy builds on the idea that the receipt of several duplicated ACKs is indicative of a lost segment

TCP Strategies

Fast Retransmit

Recall that the argument is that one or two duplicate ACKs might simply be due to out-of-order delivery, as IP is unreliable

TCP Strategies

Fast Retransmit

Recall that the argument is that one or two duplicate ACKs might simply be due to out-of-order delivery, as IP is unreliable

Three or more is taken to mean something is wrong

TCP Strategies

Fast Retransmit

Recall that the argument is that one or two duplicate ACKs might simply be due to out-of-order delivery, as IP is unreliable

Three or more is taken to mean something is wrong

If this happens, the sender should retransmit the indicated segment immediately: fast retransmit

TCP Strategies

Fast Recovery

Next, Jacobsen says do not go into slow start but do congestion avoidance: this is the *fast recovery strategy*

TCP Strategies

Fast Recovery

Next, Jacobsen says do not go into slow start but do congestion avoidance: this is the *fast recovery strategy*

We don't want slow start as the duplicate ACKs indicate that later data have reached the destination and is buffered there

TCP Strategies

Fast Recovery

Next, Jacobsen says do not go into slow start but do congestion avoidance: this is the *fast recovery strategy*

We don't want slow start as the duplicate ACKs indicate that later data have reached the destination and is buffered there

So data is still arriving (mostly) and we don't want to abruptly cut the flow by doing slow start

TCP Strategies

Fast Recovery

Next, Jacobsen says do not go into slow start but do congestion avoidance: this is the *fast recovery strategy*

We don't want slow start as the duplicate ACKs indicate that later data have reached the destination and is buffered there

So data is still arriving (mostly) and we don't want to abruptly cut the flow by doing slow start

Fast Retransmit & Fast Recovery are quite effective at getting the flow going again after a loss

TCP Strategies

Fast Recovery

Next, Jacobsen says do not go into slow start but do congestion avoidance: this is the *fast recovery strategy*

We don't want slow start as the duplicate ACKs indicate that later data have reached the destination and is buffered there

So data is still arriving (mostly) and we don't want to abruptly cut the flow by doing slow start

Fast Retransmit & Fast Recovery are quite effective at getting the flow going again after a loss

Exercise Read RFC2001 for the details we have not mentioned here

TCP Strategies

Congestion

There have been many tweaks to this basic flow control strategy

TCP Strategies

Congestion

There have been many tweaks to this basic flow control strategy

- Larger initial `ssthresh`
- Larger initial `cwnd`
- Slow start counting number of segments ACKed, not just the number of ACKs
- Treating duplicate ACKs like a timeout
- On timeout, setting `cwnd` to half `ssthresh`, not just 1 segment
- Fast recovery: wait for the ACK of the entire transmit window before entering congestion avoidance
- Many more

TCP Strategies

Congestion

Exercise Read about other strategies, such as TCP Reno, TCP Vegas, TCP New Reno, TCP Hybla, BIC, CUBIC, etc.

TCP Strategies

Congestion

And other *kinds* of congestion strategy exist and are used

TCP Strategies

Congestion

And other *kinds* of congestion strategy exist and are used

For example, BBR (specifically BBRv3) from Google is not (primarily) loss based, but develops a model of the state of the network by monitoring RTTs and the achieved bandwidth of a connection

TCP Strategies

Congestion

And other *kinds* of congestion strategy exist and are used

For example, BBR (specifically BBRv3) from Google is not (primarily) loss based, but develops a model of the state of the network by monitoring RTTs and the achieved bandwidth of a connection

It remembers and uses past behaviour as a predictor: not just the current ACK loss behaviour

TCP Strategies

Congestion

And other *kinds* of congestion strategy exist and are used

For example, BBR (specifically BBRv3) from Google is not (primarily) loss based, but develops a model of the state of the network by monitoring RTTs and the achieved bandwidth of a connection

It remembers and uses past behaviour as a predictor: not just the current ACK loss behaviour

Of course, this involves a lot of CPU cycles and could not have been done in the early days of the Internet

TCP Strategies

Congestion

And other *kinds* of congestion strategy exist and are used

For example, BBR (specifically BBRv3) from Google is not (primarily) loss based, but develops a model of the state of the network by monitoring RTTs and the achieved bandwidth of a connection

It remembers and uses past behaviour as a predictor: not just the current ACK loss behaviour

Of course, this involves a lot of CPU cycles and could not have been done in the early days of the Internet

Exercise Read about this

TCP Strategies

Congestion

Other strategies involve the routers — they are where the congestion is happening, after all!

TCP Strategies

Congestion

Other strategies involve the routers — they are where the congestion is happening, after all!

Particularly *Explicit Congestion Notification* (ECN), which aims to indicate congestion *before* it happens by routers setting flags in the IP TOS/DS header when they think congestion is imminent, so that the hosts get forewarning and can slow down

TCP Strategies

Congestion

Other strategies involve the routers — they are where the congestion is happening, after all!

Particularly *Explicit Congestion Notification* (ECN), which aims to indicate congestion *before* it happens by routers setting flags in the IP TOS/DS header when they think congestion is imminent, so that the hosts get forewarning and can slow down

Exercise Read about ECN and its use of flags in both the IP header and the TCP header

TCP Strategies

Congestion

Exercise Read about Random Early Detection/Drop (RED), which is also used in routers

TCP Strategies

Congestion

Exercise Read about Random Early Detection/Drop (RED), which is also used in routers

Exercise We use ICMP to indicate other kinds of errors, but why is it not a good idea to use ICMP when a router drops a packet due to congestion?

TCP Strategies

`tcpdump`

Exercise Use `tcpdump` to watch these strategies in operation. The `netcat` program is an easy way to set up connections and send data

TCP Strategies

Path MTU Discovery

The next strategy we have seen already: it is aimed at getting the largest segment size a connection can handle. But not too large

TCP Strategies

Path MTU Discovery

The next strategy we have seen already: it is aimed at getting the largest segment size a connection can handle. But not too large

IP layer fragmentation is expensive, so we employ path MTU discovery: but now we need to look at it from a TCP perspective

TCP Strategies

Path MTU Discovery

The next strategy we have seen already: it is aimed at getting the largest segment size a connection can handle. But not too large

IP layer fragmentation is expensive, so we employ path MTU discovery: but now we need to look at it from a TCP perspective

TCP has (potentially) more information: namely the optional MSS header sent in the setup handshake

TCP Strategies

Path MTU Discovery

We can send segments of decreasing size, starting with the minimum of the MSS of the sending interface and the MSS announced by the other end, or 536 if the other end did not give an MSS

TCP Strategies

Path MTU Discovery

We can send segments of decreasing size, starting with the minimum of the MSS of the sending interface and the MSS announced by the other end, or 536 if the other end did not give an MSS

And with the IP flag DF (Don't Fragment) set

TCP Strategies

Path MTU Discovery

We can send segments of decreasing size, starting with the minimum of the MSS of the sending interface and the MSS announced by the other end, or 536 if the other end did not give an MSS

And with the IP flag DF (Don't Fragment) set

Note the cross-layer activity here!

TCP Strategies

Path MTU Discovery

If an ICMP error “fragmentation needed but DF set” happens during a TCP connection, the congestion window should remain unchanged, but it should only resend one segment before ACKs start appearing again

TCP Strategies

Path MTU Discovery

If an ICMP error “fragmentation needed but DF set” happens during a TCP connection, the congestion window should remain unchanged, but it should only resend one segment before ACKs start appearing again

This is to reflect the fact that it’s not congestion at fault here, but we do need to back off a bit to allow ACKs to start coming through again

TCP Strategies

Path MTU Discovery

If an ICMP error “fragmentation needed but DF set” happens during a TCP connection, the congestion window should remain unchanged, but it should only resend one segment before ACKs start appearing again

This is to reflect the fact that it’s not congestion at fault here, but we do need to back off a bit to allow ACKs to start coming through again

It is recommended you try a larger MTU once in a while, e.g., every 10 minutes, as routes can vary dynamically