Routing

This is simple to implement and run, but such protocols have a *slow convergence* problem

Routing

This is simple to implement and run, but such protocols have a *slow convergence* problem

This means that if the network changes (e.g., a link is broken, or a new link is made) it takes many interchanges of information for the routers to adjust to the new routes

Routing

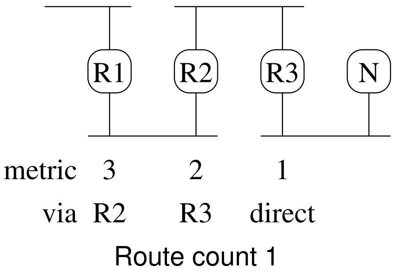
This is simple to implement and run, but such protocols have a *slow convergence* problem

This means that if the network changes (e.g., a link is broken, or a new link is made) it takes many interchanges of information for the routers to adjust to the new routes

And this can manifest in bad ways

Routing

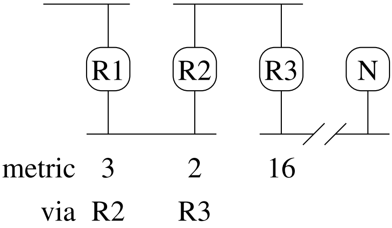
Slow Convergence



R3 knows a route to network N of hop count 1;

Routing

Slow Convergence

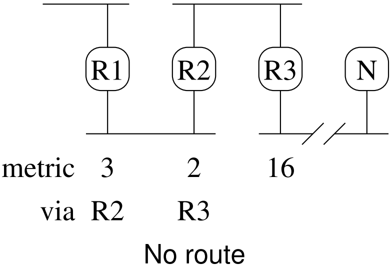


Break in route

After a break in the network R3 finds that route no longer works;

Routing

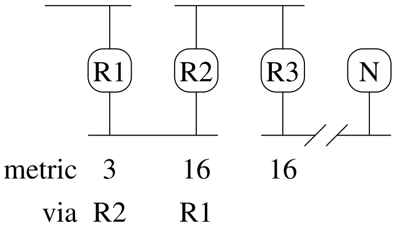
Slow Convergence



So it sends a message to its neighbours (R2) saying “no route to N”. It uses a count of 16, which is interpreted as infinity;

Routing

Slow Convergence

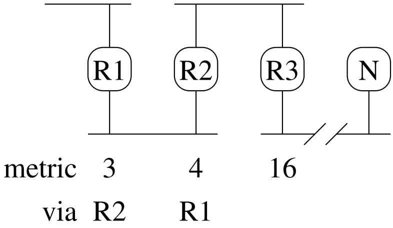


Update routing table

R2 updates its routing table;

Routing

Slow Convergence

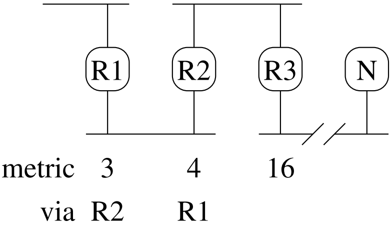


New route for R2?

But R2 also gets a periodic update message from R1 saying "route of 3 hops";

Routing

Slow Convergence

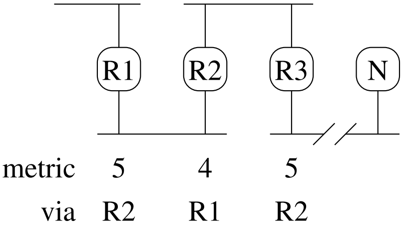


R2 update again

So R2 now thinks the best route is via R1, 4 hops;

Routing

Slow Convergence

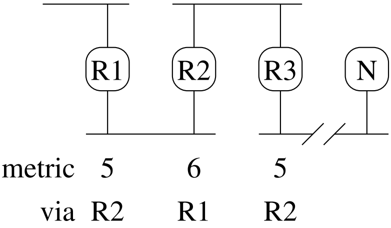


Another broadcast

And when R2 sends its periodic update message “4” to R1 and R3, R1 now thinks there is a route via R2 of 5 hops; and R3 thinks there is a route of 5 hops via R2;

Routing

Slow Convergence

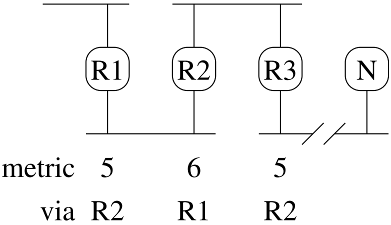


Counts grow slowly

After the next update, R2 thinks there is a route via R3 of 6 hops;

Routing

Slow Convergence

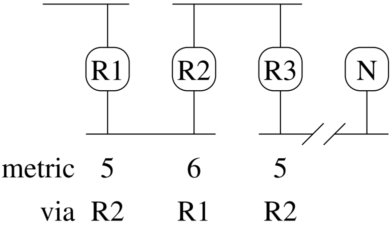


Counts grow slowly

And so on;

Routing

Slow Convergence

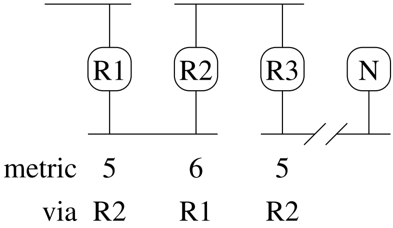


Counts grow slowly

Eventually the hop count reaches 16, i.e., no route, and so this route is dropped;

Routing

Slow Convergence

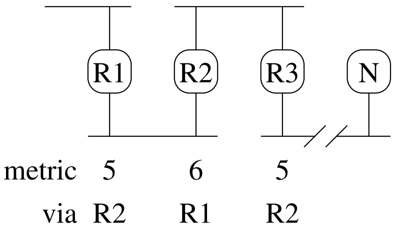


Counts grow slowly

This is called the *count to infinity* problem;

Routing

Slow Convergence



Counts grow slowly

If there was a valid route, it might take a long time to converge to that route

Routing

Meanwhile real data packets are bouncing forwards and back between the routers

Routing

Meanwhile real data packets are bouncing forwards and back between the routers

The local information that distance vector provides is not enough

Routing

Meanwhile real data packets are bouncing forwards and back between the routers

The local information that distance vector provides is not enough

RIP uses distance vector and this is a real problem for it

Routing

Meanwhile real data packets are bouncing forwards and back between the routers

The local information that distance vector provides is not enough

RIP uses distance vector and this is a real problem for it

So RIP should only used on small networks that are fairly stable

Routing

Meanwhile real data packets are bouncing forwards and back between the routers

The local information that distance vector provides is not enough

RIP uses distance vector and this is a real problem for it

So RIP should only used on small networks that are fairly stable

Link state protocols, e.g., OSPF, converge faster, but need more complicated graph traversal algorithms to determine best routes

BGP

BGP is a *path vector* variation of distance vector: this includes the path (multiple hops) to the destination, which can be used to spot the loops that lead to count-to-infinity

BGP

BGP is a *path vector* variation of distance vector: this includes the path (multiple hops) to the destination, which can be used to spot the loops that lead to count-to-infinity

ASs do not change very much so slow convergence is not such a big problem anyway

BGP

BGP is a *path vector* variation of distance vector: this includes the path (multiple hops) to the destination, which can be used to spot the loops that lead to count-to-infinity

ASs do not change very much so slow convergence is not such a big problem anyway

Exercise Read about path vector systems

BGP

BGP does have other problems, particularly authentication

BGP

BGP does have other problems, particularly authentication

Through accident or malice it is easy to trick BGP

BGP

BGP does have other problems, particularly authentication

Through accident or malice it is easy to trick BGP

For example, it would be relatively easy to get BGP to transit data through an evil third party

BGP

BGP does have other problems, particularly authentication

Through accident or malice it is easy to trick BGP

For example, it would be relatively easy to get BGP to transit data through an evil third party

Also, see the problem with the route to Youtube, earlier

BGP

Exercise Read about the 2018 hack on the cryptocurrency website `MyEtherWallet.com` that started by subverting BGP to send DNS traffic to a rogue server

Exercise Read about the BGP problem of April 2021, where Vodafone Idea (AS55410) published bad routes

Exercise Read about the proposed *Resource Public Key Infrastructure* (RPKI), RFC6810

Exercise Read about the *Mutually Agreed Norms for Routing Security* (MANRS) initiative for ISPs and routing exchange operators

BGP

Exercise Read about RIP

Exercise Read about Dijkstra's algorithm for finding shortest paths in a graph; and OSPF which uses this algorithm

Transport Layer

We now move up a layer: the Transport Layer

Transport Layer

We now move up a layer: the Transport Layer

The Internet Protocol has three main protocols that run on top of IP: two are for data, one for control

Transport Layer

We now move up a layer: the Transport Layer

The Internet Protocol has three main protocols that run on top of IP: two are for data, one for control

The data protocols are complementary

- one is fast, unreliable, connectionless: UDP
- the other is more sophisticated, reliable and connection-oriented: TCP

Transport Layer

We now move up a layer: the Transport Layer

The Internet Protocol has three main protocols that run on top of IP: two are for data, one for control

The data protocols are complementary

- one is fast, unreliable, connectionless: UDP
- the other is more sophisticated, reliable and connection-oriented: TCP

The control protocol, ICMP, we have already seen and is usually considered as part of the network layer

Transport Layer

We now move up a layer: the Transport Layer

The Internet Protocol has three main protocols that run on top of IP: two are for data, one for control

The data protocols are complementary

- one is fast, unreliable, connectionless: UDP
- the other is more sophisticated, reliable and connection-oriented: TCP

The control protocol, ICMP, we have already seen and is usually considered as part of the network layer

Other data protocols exist in this layer, but TCP and UDP are currently the important ones

Transport Layer

Ports

Both UDP and TCP use the concept of *ports*

Transport Layer
Ports

Both UDP and TCP use the concept of *ports*

On a single server machine there can be many programs running, web, email, and so on: how does a client indicate which service it wants from the server?

Transport Layer

Ports

Both UDP and TCP use the concept of *ports*

On a single server machine there can be many programs running, web, email, and so on: how does a client indicate which service it wants from the server?

And when a reply packet arrives back at a client, how does the OS know which of the many processes running on the client that packet should be delivered to?

Transport Layer

Ports

Both UDP and TCP use the concept of *ports*

On a single server machine there can be many programs running, web, email, and so on: how does a client indicate which service it wants from the server?

And when a reply packet arrives back at a client, how does the OS know which of the many processes running on the client that packet should be delivered to?

This is done by ports

Transport Layer

Ports

Both UDP and TCP use the concept of *ports*

On a single server machine there can be many programs running, web, email, and so on: how does a client indicate which service it wants from the server?

And when a reply packet arrives back at a client, how does the OS know which of the many processes running on the client that packet should be delivered to?

This is done by ports

A port is just a 16 bit integer: 1-65535

Transport Layer

Ports

Every TCP and UDP connection has a *source port* and a *destination port*

Transport Layer

Ports

Every TCP and UDP connection has a *source port* and a *destination port*

When a service starts
— i.e., a program that will deal with the service starts —
it *listens* on a port
— i.e., it informs the operating system that it wishes to receive data from packets directed to that port number

Transport Layer

Ports

Every TCP and UDP connection has a *source port* and a *destination port*

When a service starts
— i.e., a program that will deal with the service starts —
it *listens* on a port
— i.e., it informs the operating system that it wishes to receive data from packets directed to that port number

E.g., an email server may indicate it wants packets addressed to TCP port 25; a browser would listen on port 80 (and 443)

Transport Layer

Ports

The OS checks that port is not already being used by another program, and subsequently ensures packets with that destination port are sent to that service program

Transport Layer

Ports

The OS checks that port is not already being used by another program, and subsequently ensures packets with that destination port are sent to that service program

So when a TCP packet with destination port 25 arrives its data will be given to the email program

Transport Layer

Ports

The OS checks that port is not already being used by another program, and subsequently ensures packets with that destination port are sent to that service program

So when a TCP packet with destination port 25 arrives its data will be given to the email program

An analogy: a host as a block of flats. To address a letter to a specific person you need both a building address (IP address) and a flat number (port)

Transport Layer

Ports

TCP and UDP ports are entirely separate: one service can be listening for a TCP connection on a port and another service for UDP on the same port number

Transport Layer

Ports

TCP and UDP ports are entirely separate: one service can be listening for a TCP connection on a port and another service for UDP on the same port number

The OS can distinguish the two as they are port within different protocols

Transport Layer
Ports

TCP and UDP ports are entirely separate: one service can be listening for a TCP connection on a port and another service for UDP on the same port number

The OS can distinguish the two as they are port within different protocols

TCP and UDP are completely separate and do not interact at all (at the transport level)

Transport Layer

Ports

Certain *well-known* ports are associated certain services

- web server on port 80 (or 443 for a secure version)
- email server on port 25
- FTP on port 21
- Microsoft SQL server on 1433
- hundreds of others. See `/etc/services` and RFC6335

Transport Layer

Ports

Certain *well-known* ports are associated certain services

- web server on port 80 (or 443 for a secure version)
- email server on port 25
- FTP on port 21
- Microsoft SQL server on 1433
- hundreds of others. See `/etc/services` and RFC6335

A range of ports are reserved for privileged (root/administrator) programs; most are available to any program that wants to use them

Transport Layer

Ports

Certain *well-known* ports are associated certain services

- web server on port 80 (or 443 for a secure version)
- email server on port 25
- FTP on port 21
- Microsoft SQL server on 1433
- hundreds of others. See `/etc/services` and RFC6335

A range of ports are reserved for privileged (root/administrator) programs; most are available to any program that wants to use them

Typically, port numbers under 1024 are reserved for privileged programs

These associations of port numbers to services are purely convention and for convenience only: no port is special and you can run any service on any port

Transport Layer
Ports

These associations of port numbers to services are purely convention and for convenience only: no port is special and you can run any service on any port

It just means you don't have the extra problem of determining the port for, say, the web server: it is almost always 80 (or 443)

Transport Layer

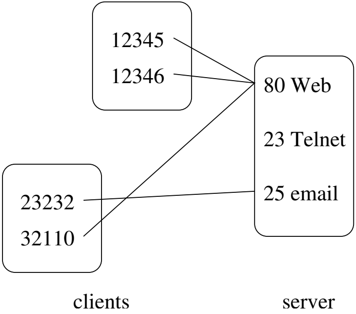
Ports

These associations of port numbers to services are purely convention and for convenience only: no port is special and you can run any service on any port

It just means you don't have the extra problem of determining the port for, say, the web server: it is almost always 80 (or 443)

You can run a web server on port 25 if you wish: you will just confuse anyone who tries to send you email

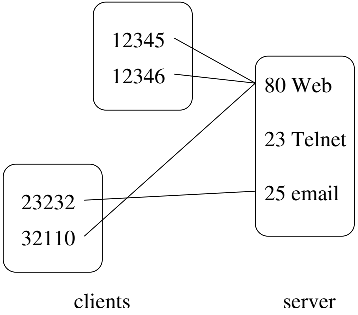
Transport Layer



Transport layer ports

Ports also enable multiple simultaneous connections between two machines, e.g., fetching several web pages

Transport Layer



Transport layer ports

Ports also enable multiple simultaneous connections between two machines, e.g., fetching several web pages

The source port (destination port on the returning packet) allows the client OS to identify which packet belongs to which client program

Transport Layer

Ports

Source ports are usually chosen afresh “at random” (usually: just increment by 1 for each time) for each new connection and are called *ephemeral* ports as they only live for the duration of the connection

Transport Layer

Ports

Source ports are usually chosen afresh “at random” (usually: just increment by 1 for each time) for each new connection and are called *ephemeral* ports as they only live for the duration of the connection

There is no technical difference between ephemeral and well-known ports, just the way they are used

Transport Layer

Ports

Source ports are usually chosen afresh “at random” (usually: just increment by 1 for each time) for each new connection and are called *ephemeral* ports as they only live for the duration of the connection

There is no technical difference between ephemeral and well-known ports, just the way they are used

The quad

source address
source port
destination address
destination port

specifies a connection uniquely: the hosts involved and the processes on those hosts

Transport Layer

Ports

The pair (source address, source port) is often called a *socket*

Transport Layer

Ports

The pair (source address, source port) is often called a *socket*

A full quad is then called a *socket pair*

Transport Layer

Ports

The pair (source address, source port) is often called a *socket*

A full quad is then called a *socket pair*

Both TCP and UDP have port fields early in their headers: this is so that the port numbers are included in the “IP header plus 8 bytes of data” that an ICMP error contains

Transport Layer

Ports

The pair (source address, source port) is often called a *socket*

A full quad is then called a *socket pair*

Both TCP and UDP have port fields early in their headers: this is so that the port numbers are included in the “IP header plus 8 bytes of data” that an ICMP error contains

Thus the OS can identify which process an ICMP belongs to

Transport Layer

Ports

The pair (source address, source port) is often called a *socket*

A full quad is then called a *socket pair*

Both TCP and UDP have port fields early in their headers: this is so that the port numbers are included in the “IP header plus 8 bytes of data” that an ICMP error contains

Thus the OS can identify which process an ICMP belongs to

And a non-initial IP fragment won’t have such identifying information, so this is why ICMPs are not generated for errors involving such fragments

Transport Layer

NAT and Ports

And ports are how a NAT firewall does its magic of matching returning reply packets to request packets

Transport Layer

NAT and Ports

And ports are how a NAT firewall does its magic of matching returning reply packets to request packets

It keeps a list of private (internal) socket pairs against public (external) socket pairs

Transport Layer

NAT and Ports

And ports are how a NAT firewall does its magic of matching returning reply packets to request packets

It keeps a list of private (internal) socket pairs against public (external) socket pairs

And this is enough to match up replies with requests

Exercise Read about *Port Address Translation*

Exercise Read about *Port Address Translation*

Exercise Sometime we wish to allow an external host to initiate a connection with a private host behind NAT. Read about *port forwarding*

Transport Layer

NAT and Ports

Exercise Read about *Port Address Translation*

Exercise Sometime we wish to allow an external host to initiate a connection with a private host behind NAT. Read about *port forwarding*

Exercise Reflect upon the idea that ports are "process addresses", namely a way to identify a particular process within a destination

UDP

We start with the *User Datagram Protocol* (UDP) as it is simpler, though historically it came along much later than TCP

UDP

We start with the *User Datagram Protocol* (UDP) as it is simpler, though historically it came along much later than TCP

UDP is the transport layer for an unreliable, connectionless protocol

UDP

We start with the *User Datagram Protocol* (UDP) as it is simpler, though historically it came along much later than TCP

UDP is the transport layer for an unreliable, connectionless protocol

Recall that “unreliable” means “not guaranteed reliable”

UDP

We start with the *User Datagram Protocol* (UDP) as it is simpler, though historically it came along much later than TCP

UDP is the transport layer for an unreliable, connectionless protocol

Recall that “unreliable” means “not guaranteed reliable”

UDP is not much more than IP with ports

UDP

We start with the *User Datagram Protocol* (UDP) as it is simpler, though historically it came along much later than TCP

UDP is the transport layer for an unreliable, connectionless protocol

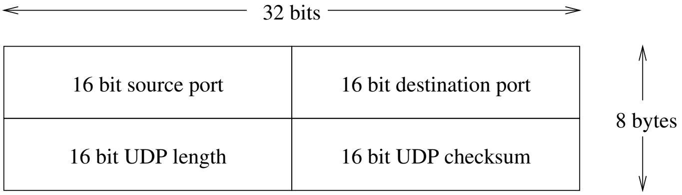
Recall that "unreliable" means "not guaranteed reliable"

UDP is not much more than IP with ports

UDP packets are typically called *datagrams* (like telegrams: simple individual messages)

UDP

Header



UDP header

UDP

Header

32 bits

16 bit source port	16 bit destination port
16 bit UDP length	16 bit UDP checksum

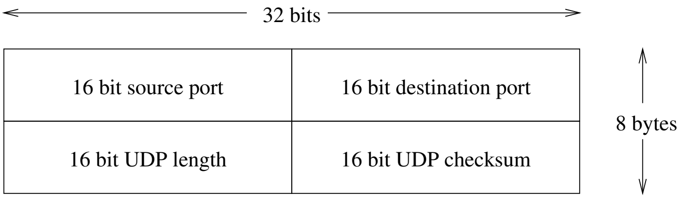
8 bytes

UDP header

- Ports: as described

UDP

Header

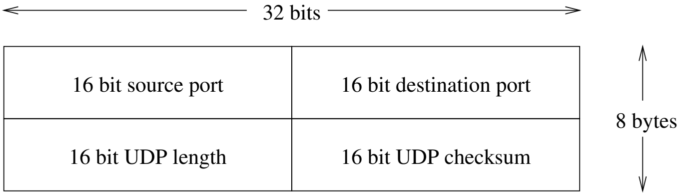


UDP header

- Ports: as described
- Length: of the entire packet, including the 8 bytes of the header: this could be deduced from the IP layer, but this keeps layer independence

UDP

Header



UDP header

- Ports: as described
- Length: of the entire packet, including the 8 bytes of the header: this could be deduced from the IP layer, but this keeps layer independence
- Checksum: of the UDP header, the data *and some fields from the IP header*

UDP

Incorporating fields from the IP header is poor design, as it ties UDP to IPv4

UDP

Incorporating fields from the IP header is poor design, as it ties UDP to IPv4

Changing the Network layer (e.g., to IPv6) involves changing the way this checksum is computed

UDP

Incorporating fields from the IP header is poor design, as it ties UDP to IPv4

Changing the Network layer (e.g., to IPv6) involves changing the way this checksum is computed

Thus adding extra complication to the v4 to v6 transition

UDP

Incorporating fields from the IP header is poor design, as it ties UDP to IPv4

Changing the Network layer (e.g., to IPv6) involves changing the way this checksum is computed

Thus adding extra complication to the v4 to v6 transition

The checksum is optional: put 0 in this field if you want to save a little time: recall UDP is unreliable!

UDP

UDP is a very thin layer on top of IP

UDP

UDP is a very thin layer on top of IP

It is as reliable or unreliable as the IP it runs on

UDP

UDP is a very thin layer on top of IP

It is as reliable or unreliable as the IP it runs on

It is just about as fast and efficient as IP, with only a small overhead (8 bytes)

UDP

UDP is widely used as it is good in a few areas:

UDP

UDP is widely used as it is good in a few areas:

- One shot applications. Where we have a single request and reply. For example, DNS

UDP

UDP is widely used as it is good in a few areas:

- One shot applications. Where we have a single request and reply. For example, DNS
- Where a fast response is required. We have no overhead in setting up a connection before data can be exchanged (see TCP). E.g., DNS

UDP

UDP is widely used as it is good in a few areas:

- One shot applications. Where we have a single request and reply. For example, DNS
- Where a fast response is required. We have no overhead in setting up a connection before data can be exchanged (see TCP). E.g., DNS
- Where speed is more important than accuracy. For example, media streaming, where the occasional lost packet is not a problem, but a slow packet is

UDP

No provision is made for lost or duplicated packets in UDP. Any application that uses UDP must deal with these issues itself, as required

UDP

No provision is made for lost or duplicated packets in UDP. Any application that uses UDP must deal with these issues itself, as required

For example, DNS over UDP sets a timer when a request is sent. If the reply takes too long in coming, assume the request or the reply was lost and resend the request

UDP

No provision is made for lost or duplicated packets in UDP. Any application that uses UDP must deal with these issues itself, as required

For example, DNS over UDP sets a timer when a request is sent. If the reply takes too long in coming, assume the request or the reply was lost and resend the request

Duplicates are not a problem with DNS

UDP

No provision is made for lost or duplicated packets in UDP. Any application that uses UDP must deal with these issues itself, as required

For example, DNS over UDP sets a timer when a request is sent. If the reply takes too long in coming, assume the request or the reply was lost and resend the request

Duplicates are not a problem with DNS

A video streamer might just patch over a lost packet with a copy of a previous packet; and so on

UDP

No provision is made for lost or duplicated packets in UDP. Any application that uses UDP must deal with these issues itself, as required

For example, DNS over UDP sets a timer when a request is sent. If the reply takes too long in coming, assume the request or the reply was lost and resend the request

Duplicates are not a problem with DNS

A video streamer might just patch over a lost packet with a copy of a previous packet; and so on

Exercise UDP is ideal for streaming video and audio, but a lot of services use HTTP over TCP. What are the advantages and disadvantages of doing this?

UDP

UDP is a widely used protocol (e.g., streaming video or audio), but we also require a reliable way of sending data

UDP

UDP is a widely used protocol (e.g., streaming video or audio), but we also require a reliable way of sending data

Thus the need for TCP