Network Layer: Virtual Circuit and Datagram Networks

Connection and Connectionless Service

A datagram network provides network-layer connectionless service, and a virtual circuit network provides network-layer connection-oriented service.

These are analogous to the transport layer services, except that:

- the service is host-to-host
- the network can't provide both, so you have to choose one or the other
- the services are implemented in the network core

Virtual Circuit Networks

These came from telecommunications – the source to destination path behaves like a telephone circuit (in performance and in the network actions along the path).

Every packet carries a VC identifier, which identifies which virtual circuit the packet is a part of.

Every router maintains state for each passing connection – when it sees a packet for a particular connection, it needs to know where to send it. It uses a forwarding table for this.

There is a setup and teardown for each call before data can flow – this creates and destroys the required state.

Resources in the links and routers (bandwidth, buffers) can be dedicated to each virtual circuit, to give predictable service / quality guarantees.

Implementation

A virtual circuit consists of:

1. A path from source to destination

2. VC numbers, one for each link along the path
    - The VC number can be changed by the router at each link, with the new value coming from the path.
    - If we had a single VC number for each circuit, then we have to make sure they're globally unique. By making each number specific to a specific link, we're able to reuse numbers at different links, and our numbers can be smaller (only need enough numbers to identify the maximum number of circuits through one link).
3. Entries in forwarding tables in routers along the path

Forwarding Table

A forwarding table maps (VC number, incoming interface) pairs to (VC number, outgoing interface) pairs.

- making IDs local is more efficient and also simpler to allocate – decision is made between two routers
    - using one number across all the hops takes a lot more co-ordination
    - we would need globally-unique IDs, which is where we started
        - and who would allocate these? A central server?
    - instead we can just use enough numbers to cover the number of flows through the specific router
    - smaller number means we can use fewer bits to represent it – save space/state
        - these go in every header, remember!
        - this efficiency is a key difference between virtual circuit and datagram
            - though it does mean you need to change the ID in the header at each packet

Signaling Protocols

- used to setup, maintain, and teardown VC
- used in ATM (asynchronous transmission mode?), frame relay, x.25
- can allocate resources as well

Datagram Networks

- no call setup at network layer
- routers don't keep state about end-to-end connections
  - no network-level concept of a connection
  - doesn't matter if a router fails – other routers will just avoid the dead link
    - what would we do in VC?
- packets forwarded using destination host address
  - need global scope
  - packets between the same source-destination pair may take different paths and arrive out of order

Datagram Forwarding Table

- each entry contains an address range and an output to send those to
- usually a default rule for packets that don't match the others

Longest Prefix Matching

- provide a bunch of prefixes, the longest matching one determines the destination
  - e.g. if the destination address matches prefixes `010` and `0100`, the interface specified for `0100` is used because `0100` is longer than `010`

Datagram vs. VC

- VC/ATM not as widely used, partly because it was competing with an already establish technology, and partly because of economic complexity linked to guaranteed quality of service (who gets it, do you charge for it, etc.)

VC

- VC has guaranteed service, can satisfy strict timing and reliability requirements
- simple end systems (e.g. telephones), with complexity inside network
- once the connection is set up, forwarding is much faster, because you don't need to do the prefix matching

    - constant-size indices are simpler, and can be implemented in software
- if a router dies, it's complicated to re-establish the connection

Datagram

- Datagram has no guaranteed service
    - but end systems are complex, and can adapt to the unreliable channel, performing control and error recovery
    - then routers can be very simple
- router dying is more straightforward to overcome

Router Architecture Overview

2 key functions:

- running routing algorithms/protocol
- forwarding datagrams from incoming to outgoing link

Typically the routing processor pushes configurations to hardware for each input port, so that the ports don't have to check with the processor every time

```
* routing processor is control plane

    * this is software

* everything else is data plane

    * this is hardware
```

Input Port Functions

- physical layer -> link layer -> network layer
- decentralised switching
    - input ports have memory with a forwarding table, which is updated every now and then by the routing processor
- goal is to have input port processing at line speed (speed of the physical layer)
    - otherwise you get queueing, possible congestion and packet loss

Switching Fabrics

Shared Memory

- simplest
- cheapest
- can be done in software on a PC
- involves copying
  - input port copies into memory
  - output port copies from memory
  - only one thing can use the system bus at any one time
  - this can be quite slow, which lowers your throughput

Bus

- shared bus from input port memory to output port memory
- bus contention: switching speed limited by bus bandwidth (e.g. 40 Gb/s)
- bus requires exclusive access (only one device transmits on it at a time)

Crossbar

Aims to alleviate the problem with the bus of only one thing being able to transmit at once. Transfers that don't conflict are allowed in parallel.

Some advanced designs split datagrams into fixed length cells, making the switching and routing through the fabric more deterministic. Some of them basically use ATM.

Output Ports

If the switching can happen faster than the transmission of packets, then packets buffer.

You may need to think about a scheduling algorithm – could use FIFO, or could possibly prioritise certain packets, e.g.:

- control packets for routers
- low-latency content like VOIP or video games

How Much Buffering?

- RFC 3439 rule of thumb: typical RTT multiplied by the link capacity

A more recent recommendation says to divide this value by `sqrt(N)`, where `N` is the number of flows.

These are both worked out from analysing existing networks.

Input Port Queueing

If the switching fabric is slower than the combined speed of the input ports, you can get queues.

Also can get head-of-the-line blocking, where a packet that has to wait (because the output is in use) blocks one that wouldn't have to wait.

- Can solve this using a scheduling algorithm at the input ports.