1

PRINCIPLES OF COMMUNICATIONS

Routing

Finding path ~~for~~ from source to destination in a network. Issues include:
- ↳ what route?
- ↳ is that the shortest route
- ↳ what happens if link goes down?
- ↳ Multipath routing (for mobile wireless link)
- ↳ latency, bandwidth etc

① Interdomain Routing: socioeconomic issue – comms between different stakeholders
② Multicast Routing
③ Routing in telephone networks → often just random routing
  ↳ useful also for load balancing
④ Routing for mobile hosts.

Packet Switching

(1) Routing protocol sets up a (2) routing table in routers and switch controller
- ↳ node makes a local choice depending on global topology and state
- ↳ Separation between control and data view of network (and group addressing). We may have a centralised system
  - ↳ inherently large – can't topologically sort – if we can sort hierarchically then aggregate route entries
  - ↳ Dynamic traffic conditions
  - ↳ hard to collect (failures, etc) → restore are fast – do we wait or find a new path
- ↳ Must intelligently summarize relevant information

Requirements

① Minimize routing table space
- ↳ fast to lookup
- ↳ less to exchange

] expensive

② Minimize number and frequency of control messages – dealing with lots of changes
③ Robustness, avoiding:
- ↳ Black Holes: Islamabad example where YouTube ISP was made by ~~mod~~ to redirect to them – hence, best route was to go via this hence entire world followed this path
- ↳ Loops: TTL is a last resort; especially with multiclass traffic
- ↳ Oscillations:

④ Use optimal path

Features: Goal: maximise throughput, subject to min delay and cost.
① Packets
② Topology is complicated
③ Many provides
④ Traffic sources are bursty
⑤ Traffic matrix is unpredictable and large :- traffic is more important than latencies

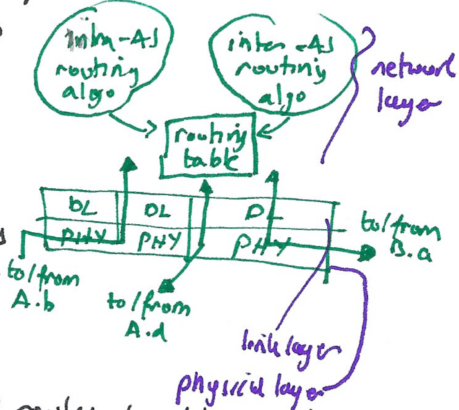
Routing Model

(1) Dynamic routing and (2) Intra and Inter AS – AS = locus of admin control
- → each ISP is an AS
- ↳ autonomous systems
- ↳ IGPs inside ASs: RIP, OSPF, HELLO
  - RIP ← oldest
  - OSPF ↳ bad
  - ↳ distribute cost of paths

Interior Gateway Protocols (IGPs)

② ↳ Exterior Gateway Protocols (EGPs) for inter AS routing.
↳ EGP, BGP-4

Gateways
① perform inter-AS routing amongst themselves
② perform intra-AS routing.



Intra-AS Routing Requirements

① Scale for size of AS
② Different requirements on routing convergence after topology changes
③ Operational / Admin / Management (OAM) complexity
④ Traffic Engineering Capabilities

Inter-AS Routing Requirements

① should scale for global internet size
↳ reachability not optimality — lots of possible goals
↳ latency, throughput,
↳ address aggregation techniques
↳ Allow flexibility in topological structure.
② Allow policy-based routing between autonomous systems.
↳ In case of routing, options include advertised AS-level routes to address prefixes
↳ Fully distributed routing is the only possibility
↳ Extensible

Basic Dynamic Routing Methods

① Source Based: source gets a map of the network - IP allows that to occur
↳ source finds route and (1) signals route-setup
or (2) encode route into packets.
↳ But, can have attacks on the way
② Link State Routing (per link info) → centralized
↳ get map of network at all nodes and find next-hops locally
③ Distance Vector (per node info)
↳ at every node, set up distance signposts to destination nodes
↳ Look at neighbours signposts → BGP

Choices

① Centralized?: simpler but prone to failure and congestion
↳ need very reliable central system
② Source-based vs hop-by-hop.
↳ more expensive + packet header size
↳ Intermediate: loose source route
③ Stochastic vs deterministic
↳ spreads load, avoiding oscillations but misorders
④ Single vs multiple path
⑤ State-dependent vs state-independent ⇒ routes depend on current network state?

Central Control over Distributed Routing (fibbing)

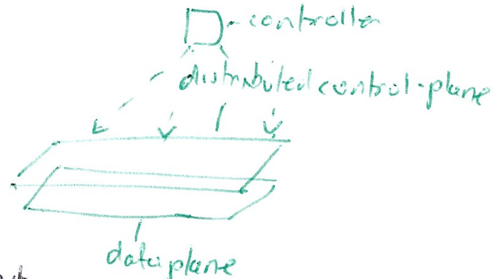
SDNs (software defined networks) :- latency can be very low using cut-through switches, also single managment area. Effectively using remote control switching. Change the table to additional entries to do what they want for entries ⇒ if what they define breaks, it returns to default. Effectively a hybrid of link state and centralised controller

	Traditional	SDN	fibbing
Manageability	low	high	high
Flexibility	low	highest	high
Scalability	by design	ad hoc	by design
Robustness	high	low	*high

↳ central system adds security
~ same as SDN
fibbing is a hybrid SDN architecture
{ per destination full control
{ some functions are distributed

* thanks to partial distribution - fallback system

③ SDN does three things: (1) computes paths, (2) derives FIB entries, (3) installs FIB entries
↳ effectively allows you to control the switches



Fibbing centralizes only high-level decisions
↳ controller:
↳ (1) computes paths
↳ distributed control-plane:
↳ (1) computes FIB entries
↳ (2) install FIB entries

To control the IGP output, the Fibbing controller inverts the shortest-path function.

input: weighted topology → function: shortest-path computation → output: forwarding plans

output format determined by operators

Flexibility

Operators can modify shortest path outputs by injecting information as fake nodes and links to the IGP control-plane
↳ any set of forwarding DAGs can be enforced by Fibbing through:
↳ ① fine-grained traffic steering – middleboxing
↳ ② per-destination load balancing – traffic engineering
↳ ③ backup paths provisioning – failure recovery

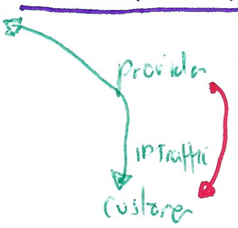
Scalability

Fibbing controller says algorithms to be run in sequence
↳ merger iteratively merges fake nodes – programs multiple next-hop changes with a single fake node
Does not impact IGP convergence – achieves fast forwarding changes

Robustness

Improves it by relying on underlying IGP which: → re-converge quickly
↳ provides fast failure detection and control-plane sync
↳ supports fail-open and fail-close semantics
↳ survive replica failures with no impact on forwarding

INTERDOMAIN ROUTING



customer pays provider for access to the internet.

peers provide transit between their respective customers. They do not provide transit between peers. Allows connectivity between customers of Tier 1 providers

peer	don't peer
reduces upstream transit costs	would rather have customers
increase upstream performance	Peers usually competition
Only way to connect customers to Tier 1	Relationships require periodic renegotiation ⇓ can be messy negotiations

Forwarding: determine next hop → always works
Routing: establish end-to-end path
↳ can be badly broken
↳ may not always find a path

④ Forwarding tables used to implement routing:

↳ Statically :- world manually configures routes : Admin manually configures forwarding table entries - mostly at the edges

↳ Dynamically : routers exchange reachability information to compute best routes

↳ Interior Gateway Protocol (IGP)

↳ metric based : OSPF, IS-IS, RIP, EIGRP

↳ Exterior Gateway Protocol (EGP)

↳ policy based : BGP

Link State	Vectoring
· Topology information flooded within the routing domain. · Best end-to-end paths computed locally at each router · Best paths determine next hop · Based on minimizing some notion of distance · Works only if policy is shared and uniform	· Router knows little about network topology · Best next-hops chosen by each router · Best paths result from composition of all next-hop choices · No notion of distance · No uniform policies

IGP = OSPF / IS-IS (Link State)

IGP = RIP, EGP = BGP (Vectoring)

Routing computation distributed among routers within routing domain

Autonomous Routing Domains (ARD) : collection of physical networks glued together using IP that have a unified administrative routing policy.

↳ AS is an ARD assigned an Autonomous System Number (ASN) - 32 bit values ⇒ these can be shared

↳ ARDs use static routing

BGP-4 : de-facto EGP today.

↳ ① Establish TCP session

↳ authentication

↳ certification

↳ ② Exchange active routes

↳ advertise what's reachable + border router

↳ ③ Exchange incremental updates

① eBGP (external) in a different Autonomous system

② iBGP (internal) in same Autonomous system.

↳ iBGP mesh does not ~~scale~~ scale

Route Reflectors

Can pass ~~it~~ on iBGP updates to client - passes through only best routes, avoiding loops

BGP Confederations : has multiple internal ASs that present one external AS

Messages

↳ ① Open

↳ ② Keepalive

↳ ③ Notification

↳ ④ Update ⇒ announcement = IP_prefix + attributes_values

Attributes including :

↳ AS_PATH

↳ NEXT_HOP

↳ MULTI EXIT DISC

↳ LOCAL PREF

↳ COMMUNITY

↳ ORIGIN

↳ CLUSTER LIST

Route Selection

- ↳ ① Highest Local Preference — used only in iBGP
- ↳ ② Shortest ASPATH
- ↳ ③ Lowest MED multi-exit descriminator
- ↳ ④ iBGP < eBGP
- ↳ ⑤ Lowest IGP cost to BGP egress
- ↳ ⑥ Lowest router ID] break all ties

(② – ⑥: traffic engineering)

BGP Route Processing

The process is as follows:

Receive BGP updates → Apply important policies (special rules) → Best Route selection → Best route table → Export policies → BGP update

Best route table ↓ IP forwarding table ↓ propagate to clients

Next hop attribute: if crosses AS boundary, NH attribute is changed to IP address of border router that announced the route

EGP joins to IGP by connecting forwarding tables and EGP tables.

Enforcing Customer/Provider and Peer/Peer relationships

① Enforce transit relationships - output route filtering

② Enforce order of route preference - ↳ provider < peer < customer

~~RSA~~

BGP Communities: has a community value of 32 bits

- ↳ First 16 bits is ASN
- ↳ Next 16 is community number

] used for signalling within and between ASes

Community Attribute = list of community values - one route can belong to multiple communities

- ↳ imports: customer routes, peer routes, provider routes
- ↳ exports: to customers, to peers, to providers

Traffic Engineering with BGP

Inbound	Outbound
① Filter outbound routes ② tweak attributes on these routes to influence ~~best~~ neighbour's best route selection.	① Filter inbound routes ② Tweak attributes on inbound routes to influence best route selection

AS has more control over outbound traffic

- ↳ don't accept BGP ASPATHS that would lead to a loop
- ↳ traffic often follows the ASPATH but sometimes doesn't
- ↳ we can implement backup links with local preference for outbound traffic and inbound traffic
- ↳ we ASPATH padding to make the course less good by making it look longer but this does not always work
  - ↳ customer routes + local preference > ASPATH length
  - ↳ can fix this with a community attribute

Hot potato routing means we go for the closest egress point

↳ this can have many issues though ⇒ can have Multi-Exit Discriminator Attribute (MEDs), hence provider network as far as possible

⑥

Route Pinning Example

original goal was just reachability and loop free
↳ (i) Couldn't optimise for a metric
↳ (ii) couldn't scale to large scale
↳ traffic remains pinned to backup if disaster strikes
N.B BGP not guaranteed to converge on stable routing ⇒ leads to livelock
↳ not guaranteed to recover from network failure
For static semantics, BGP policies solves the Stable Paths Problem.
↳ dynamic semantics, BGP solves SPVP
↳ Simple Path Vector Protocol = distributed algorithm for solving SPP

Stable Paths Problem

↳ graph of nodes and edges
↳ origin nodes
↳ For non-origin node, set of permitted paths to origin
↳ ranking of permitted paths
~~↳ origin is n~~
↳ solution does not represent a shortest path tree or spanning tree
↳ can have multiple solutions ⇒ results in route triggering
↳ bad gadget is where there is no solution

Internet Growth Trends

Large BGP tables is an issue - causes a large problem and slows things down.
↳ goals are ① fast convergence
② minimal updates
③ path redundancy
Can have bad apples ⇒ leads to local traffic meaning people have to recalculate paths - reduce affect of this by squashing updates.
↳ ① Rate limiting on sending updates - 30 seconds
↳ ② Route Flap Dampening - punishment system
↳ (1) rate limiting dampens some of oscillations inherent in a vectoring protocol.
↳ (2) Routes given penalty for changing - route is dampened which decays exponentially. If penalty goes below reuse limit, then announced again
↳ applied only on eBGP inbound only
↳ however, lots and lots of BGP updates:
↳ ① Misconfiguration
↳ ② Route flap dampening not widely used
↳ ③ Software bugs
↳ ④ BGP exploring alternate paths

MEDs export internal instability

Multicast Routing

↳ packet sent by any member of a group are received by all.
Multicast Group : associates senders and receivers
↳ sender does not know receivers identities
↳ has its own class D address which send to and receives request packet from that address.
↳ dynamic directory service associates the two
↳ Issues
↳ ① Currently active groups
↳ ② How to express interest in joining
③ Discover receivers in a group
④ Delivering data to members in group

⑦ <u>Ring Search (Expanding)</u>

Method of using multicast groups for resource discovery
↳ reaches all receives - discovering local resources first

Multicast Flavours
↳ Unipoint : point to point
↳ Multicast : point to multipoint — simulate by set of point to point unicasts
multipoint to multipoint — simulate by set of point to multipoint

(shortest-path tree)
Form multicast tree, routed at the sender

Issue in Wide-Area Multicast - exploit LAN's broadcast capability
↳ does LAN contain members for a given group
↳ what MAC address corresponds to IP address
↳ there's a well known translation ~~table~~ algorithm => no need for a translation table

each endpoint is a router => uses IGMP to get members in LAN

↳① sources join and leave dynamically
↳② leaves of tree members of broadcast LAN
↳③ want receiver to join and leave without notifying sender
↳Goal: distribute packets coming from sender to all routes on path from group member

<u>Group Management Protocol</u> : detects if LAN has any members for a particular group. (If no members, prune shortest path tree for that group by telling parent).
↳① Router periodically broadcasts query message
↳② Hosts reply with list of groups

Pruning
router tells tree parent to stop forwarding
↳ can be associated with multicast group or with source and group

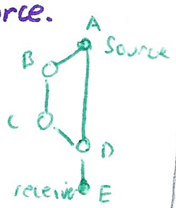
<u>Solution</u>

Simplest: flood packets from source to network ⇒ if router not seen it forward to all interfaces except incoming one

Clever: reverse path forwarding :- ① forward packet from S to all interfaces iff packet arrives on the interface that corresponds to shortest path to S.
↳ need to know shortest path -> need routing table.
② No need to remember past packets
↳ doesn't work if routes do not support multicast
③ Don't always need to forward all packets

Better: Don't send packet downstream if not on shortest path from downstream router to the source.

↳ can rejoin though (if C wants to rejoin)
↳ IGMP lets C know of host's interest
↳ ~~B sen~~ C sends join message to B which propagates from A



<u>Multicast Routing Protocol</u>

Interface on shortest path to source depends on whether the path is real or virtual

But need to discover shortest paths only taking into account multicast capable routes
↳ DVMRP - Distance Vector Multicast Routing Protocol

Used in conjunction with (1) flood and prune, (2) reverse path forwarding, (3) Explicit join messages to reduce join latency

<u>MOSPF</u> : routers flood group membership information with LSA
↳ each router independently computes shortest path tree with only multicast-capable routes
(no need to flood and prune
↳ Complex: → ① Must interact with external and summary records
↳② Need storage per group per link
↳③ Need to compute shortest path tree

<u>Core Based Trees</u> : coordinate multicast with core router: (1) host sends join request to core router, (2) routers along path mark incoming interface for forwarding. Named as rendezvous point - periodically sends 'I am alive' messages. Leaf routers set timer on receipt - if timer goes off, send request to another rendezvous point.

8

Routing for mobile hosts : issues

① Location
② Routing

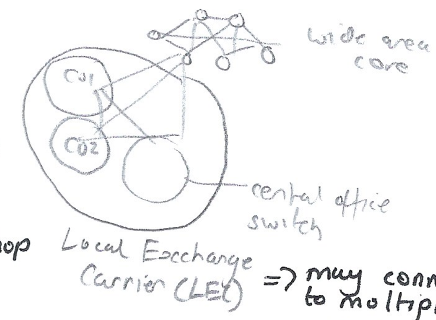
Cellular routing ↗ if you reduce size of cell, bandwidth increases

↳ System knows which cell you are seen in => send message in the backbone
↳ Each cell phone has a global ID that it tells MTSO when turned on
↳ remote MTSO tells home MTSO then to closest base => new MTSOs added as load increases
↳ mobile routing in the internet is very similar to mobile telephony but outgoing traffic does not go through home
↳ registration packets instead of slotted ALOHA - passed onto home address agent.
↳ Old care-of-agent forwards packet to new care-of-agent

Problems
① Security
② Loops

Routing in telephone networks

Old was lines to every town and would ring every town

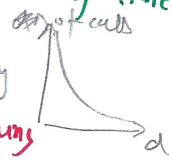


=> may connect to multiple cores

Algorithm: ① If endpoints within same CO, directly connect
② If call is between COs in same LEC, use one-hop path between COs
③ Else send call to one of the cores
④ Only decision at toll switch
↳ max two-path hop] - need to decide which two-~~path~~ hop path

Features of telephone network routing
① Stable Load
② Reliable Switches
③ Single organization controls entire core
④ Very highly connected network
⑤ Connections require resources

Statistics
↳ Poisson call arrival - independence assumption (people make calls independently)
↳ Exponential call holding time => long calls less likely
↳ not true sadly



Goal: minimise call blocking

Simplicity - historical necessity, but requires:
↳ (1) reliability in every component
↳ (2) logically fully-connected core.

Dynamic Nonhierarchical Routing

- leads to metastability : simplest core routing protocol - accept call if one-hop path is available, else drop.

↳ divides day into around 10-periods (every ~~week~~ week)
↳ in each period, each tollswitch assigned primary one-hop path + alternatives
↳ overflow to alternative if necessary
↳ drop only if alternate paths are busy
↳ rely on accurate predictions

↳ burst of activity can cause network to enter metastable state
↳ high blocking prob even with a low load
↳ Removed by trunk reservation
↳ Trunk Status Map Routing - updates measurements once an hour

Real-time Network Routing

↳ Each tollswitch maintains list of lightly loaded links
↳ Intersection of source and destination lists gives set of lightly loaded paths
] no centralized control

Dynamic Alternative Routing.

↳ whenever link is saturated, use alternative node (tandem)

Fixed Tandem
For any pair of nodes, assign fixed node k as tandem
↳ needs careful traffic analysis
↳ inflexible during breakdowns and unexpected traffic at tandem

(9)

Sticky Random Tandem

If no free circuit along (i,j), a new call is routed through a randomly chosen tandem k → as long as it does not fail
↳ if fails, call is lost and new tandem is selected

Decentralized and flexible with no pre-analysis of traffic required.

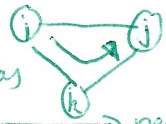
$p_k(i,j)$ = proportion of calls between i and j which go through k

$q_k(i,j)$ = proportion of calls that are blocked

$$p_a(i,j)\,q_a(i,j) = p_b(i,j)\,q_b(i,j)$$

Assign different frequencies to different tandems

Trunk Reservation: tandem accepts to forward calls if it has free capacity more than R.



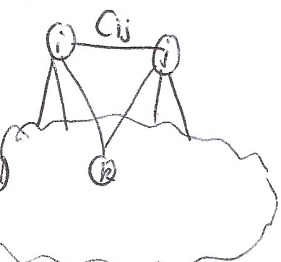
→ penalize two link calls, at least when lines are busy

Erlang's Bound: node connected to C circuits
Arrival: poisson with mean v

Expected value of i-th call blocking $E(v,C) = \frac{v^C}{C!}\left[\sum_{i=0}^{C} \frac{v^i}{i!}\right]^{-1}$ — don't memorise this.

Max-Flow Bound

Mean load $= v_{ij}$



$$E\left[\sum_{i<j} n_{ij}(t)\right] \leq f(v(t))$$

f is solution to LP: $\max \sum_{i<j}\left(x_{ij} + \sum_{j,i\neq k} x_{ikj}\right)$

↳ traffic > capacity for some links
↳ not always feasible to find set of tandems
↳ Greedy Algorithm (for $p < \frac{1}{3}$, successful with p approaching 1)
↳ (i) no saturated links → DONE
↳ (ii) saturated link and good triangle (one saturated, two not)
↳ DONE
↳ Add good triangle to list
↳ (iii) Saturated link and no good triangle → fail

Extension to DAR:
↳ ① n-link paths
↳ ② Multiple alternatives
↳ ③ Least-busy alternative
↳ ④ Repacking – call in progress can be rerouted

tradeoffs between overhead for ① stability and ② simplicity for unfairness

Flow Control Problem

↳ sender needs to match receivers rate
↳ occurs at transport or datalink layer

too fast = buffer overflow
too slow = waste time

Open Loop Flow Control

↳ Open Loop: ① source describes desired flow rate, ② network admits call, ③ source sends at this rate

① Call Setup
↳ (1) network prescribes parameters
↳ (2) user chooses parameter values
↳ (3) network admits or denies call

② Data transmission
↳ (1) User send within parameter range
↳ (2) Network polices the user
↳ (3) Scheduling policies gives the user QoS

Problem 1: choosing descriptor at source
↳ envelope that constrains worst case behaviour, used as:
↳ (1) basis for traffic contract
↳ (2) input to regulator and policer

10

Descriptor requirements:
- (1) Representativity - adequately describes flow
- (2) Verifiability
- (3) Preservability
- (4) Usability

examples: (1) time series of interarrival times (1, 2, !4), (2) peak rate (!1, 2, 3, 4), (3) average rate, (4) Linear Bounded Arrival Process

PEAK RATE

(2) Highest rate at which source can send data
- two ways to compute:
  - min interpacket spacing for networks with fixed-size packets
    - Regulator: timer set on packet transmission and if expires, send packet
  - highest rate overall packets of particular interval for networks with variable size packets
  - NB it is sensitive to extremes

(3) AVERAGE RATE

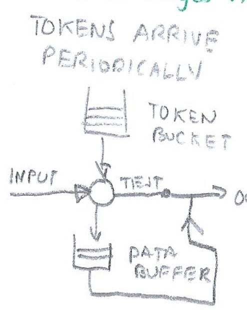
Rate over some time period (window) hence less susceptible to outliers
- JUMPING: over ~~consequetive~~ consecutive intervals of length ~~t~~ t, only a bits sent
  - regulator reinitialized every interval
- MOVING: over all interval of length t, only a bits sent
  - regulator forget packet sent more than t seconds ago

(4) Linear Bounded Arrival Process

: (1) Source bounds - # of bits sent in any time interval by a linear function of time, (2) # of bits transmitted in any active interval of length $t < rt + s$
- r = long term rate
- s = burst limit

] insensitive to outliers

- Leaky bucket is a regulator for LBAP, token bucket fills up at a rate and larges # tokens < s



Sum of token and data bucket is what matters

- Choosing parameters: tradeoff between r and s (and s (data, bucket size))
  - minimal descriptor costs less and doesn't simultaneously have smaller r and s
  - Choosing: (1) Keep loss rate same:
    - as s↑, r↓. For each r, we have at least s
    - Then choose the knee of the curve

LBAP is popular → sort of representative, verifiable, sort of preservable, sort of usable.
- NB struggles with multiple time scale traffic

Closed Loop: monitor available bandwidth ⇒ Generalized Processor Sharing and adapt to it. Can lead to loss / delay - normally if can't describe traffic

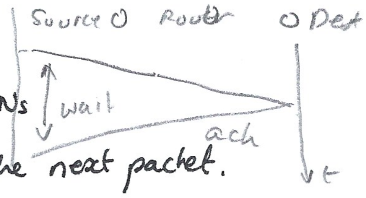
Taxonomy

- First Generation: ignore network state and match receiver
- Second Generation: responsive to state with three choices: (1) Explicit or implicit state measurement, (2) Control (flow control window size or rate), (3) Point of control (endpoint or within network)
  - Explicit: network tells source its current rate
  - implicit: endpoint figures out rate by looking at network
  - largest number of packets outstanding
  - If endpoint has sent all packets, it must wait ⇒ slows down its rate
  - Hence have transmission window (error control and flow control window)
  - window vs rate: window has no need for timer and is self-limiting; rate has better control

Hop-by-hop vs end-to-end
- easy to implement with 1st gen flow control at every link → simpler, distributes overflow → better control
- sender matches all servers on path.

(11)

On-Off



Receiver gives ON and OFF signals - used in serial lines and LANs

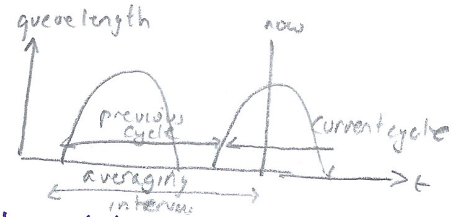
Stop and wait: (1) Send packet and (2) wait for ack before sending the next packet.

Static window: send at most one packet per RTT - works well if b and R are fixed
↳ but need to adapt window

Set window size: → bottleneck service path = b pkts/sec
RTT = R
flow control window = w
Sending rate = w/R
w/R > b ⇒ w > bR ⇒ OPTIMAL WINDOW SIZE

DECbit Flow Control

Intuition: → every packet has a bit in header
↳ intermediate routers set bit if queue has built up ⇒ source window is too large
↳ sink copies bit to ack
↳ if bit set, source reduces window size
↳ in steady state, oscillate around optimal size



Router Actions

① Measure demand and mean queue length of each source
② Computed over queue regeneration cycles
] balance between sensitivity and stability

if mean queue length > 1.0, set bit on sources whose demand exceeds fair share
if > 2.0, set bit on everyone

Source Actions

↳ Keep track of bit and can't take control actions too fast
↳ wait for past change to take effect
↳ measure bit of past + present window size
↳ if > 50% set, decrease window, else: increase
↳ additive increase, multiplicative decrease

Evaluation: works with FIFO but requires per-connection state (demand). Works with software but assumes cooperation.

TCP Flow Control: implicit with dynamic window. Very similar to DECbit:

↳ ① no support from routers
↳ ② increase if no loss
↳ ③ window decrease on timeout
↳ ④ additive increase, multiplicative decrease

SLOW START: window starts at 0, increases exponentially (doubles every RTT) for a while then linearly
each = window increase by 1
each = window increase by 1 (increases by 1 every RTT)
CONGESTION AVOIDANCE

ssthresh is called slow start threshold - contains window size in case of loss; loss detected by duplicate ACK and timeout

↳ DEPENDS ON VERSION OF TCP → Tahoe vs Reno

↳ NB TCP uses implicit measurement of congestion → operates at the cliff

TCP Vegas

$E_{throughput}$ = transmission_window_size / prop_delay (known) - measure smallest RTT
↳ can also measure actual throughput ⇒ measure difference and adjust

NETBLT

Rate-based flow control scheme, separating error control and flow control. → losses + retransmissions don't affect flow rate. Application data sent as series of buffers (at particular rate) ⇒ Rate = (burst size + burst rate)
↳ if received rate > sending rate, increase sending rate

(12)

Packet Pair

Improves basic ideas of NETBLT : → better measurement of bottleneck
↳ control based on prediction
↳ finer granularity

Spacing between packets at receiver = 1/ slowest server (Assuming bottleneck serve packets in round robin order)

ACKs give time series of service rates in the past
↳ use this to predict next rate
↳ Exp averager with fuzzy rules to change averaging factor

Predicted rate feeds into flow control equation

I = source rate

X = #packets in bottleneck buffer
S = #outstanding packets
R = RTT
b = bottleneck rate

$X = S - Rb$

$I(k+1) = b(k+1) + (\text{setpoint} - x)/R$

Hybrid Flow Control

↳ source gets minimum rate, but can use more
↳ BUT have problems of both open loop and closed loop

Design Goals
① Stability - guarantee it
② Transient Response → short settling time, small overshoot
③ Steady-state error (small)
④ Robustness
↳ (i) Disturbance rejection
↳ (ii) Sensitivity ← Low

FEEDBACK CONTROL THEORY

Why? Allows for interaction with physical environment. Want QoS guarantees especially in open, unpredictable environments

Control: applying input to cause system variables to conform to desired values called the reference

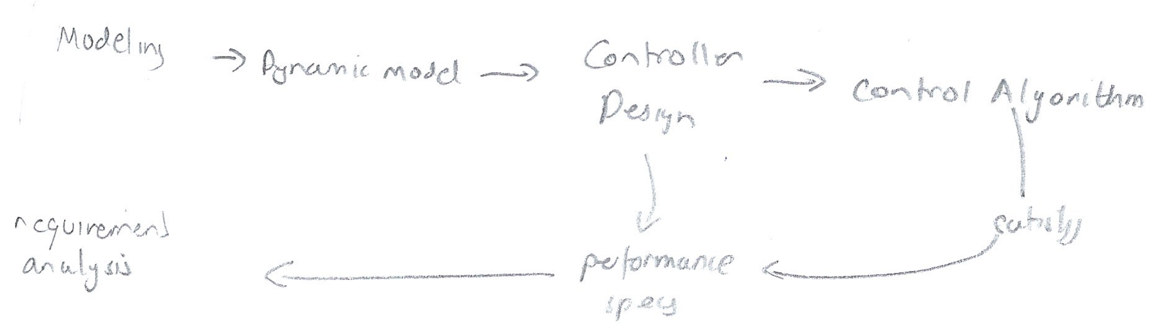
Open-Loop Control: compute control input without continuous variable measurement (but need to know everything accurately and things not to change)

Close-loop Feedback Control: measure variables and use to control input → more complicated
↳ continuously measure and correct. Don't need to know everything and things can change.

Advantages of Feedback Control Theory

① Adaptive Resource Management Heuristics: Laborious iterations for everything
② Ability to handle feedback → it's good at bad conditions, less good at okay conditions
③ Feedback Control Theory ⇒ systematic approach

Control Design Methodology



System Models

↳ ① Linear vs non-linear
↳ ② Deterministic vs Stochastic
↳ ③ Time-invariant vs Time-varying
↳ ④ Continuous time vs Discrete time
↳ ⑤ System ID vs First Principle

Dynamic Model

Characterise relationships using differential equations in either time domain or Frequency domain ⇒ Fourier Transform to go between

Controller definition:

$$a_2 \ddot{y}(t) + a_1 \dot{y}(t) + a_0 y(t) = b_1 \dot{u}(t) + b_0 u(t)$$

Transfer function (f domain)

$$G(s) = \frac{b_1 s + b_0}{a_2 s^2 + a_1 s + a_0} = \frac{c_1}{s - p_1} + \frac{c_2}{s - p_2}$$

(13)

Model Differential Equation

U = utilisation

Error : $E(t) = U_s - U(t)$

$$U(t) = \int_{s=0}^{t} (R_a(s) - R_c(s))\,ds$$

N.B. Convolution is very simple, just multiplication in the frequency domain

Laplace Transform

- similar to Fourier Transform
  - ↳ translation from time domain to s (frequency)

$$F(s) = L[f(t)] = \int_{0^-}^{\infty} f(t)e^{-sp}\,dt$$ where $s = \sigma + i\omega$ (complex variable)

$$a_2\ddot{y}(t) + a_1\dot{y}(t) + a_0 y(t) = b_1\dot{u}(t) + b_0 u(t)$$

$$\Leftrightarrow Y(s) = \frac{b_1 s + b_0}{a_2 s^2 + a_1 s + a_0} \cdot U(s)$$

Examples

Impulse : $f(t) = \delta(t) \Leftrightarrow F(s) = 1$

Step signal: $f(t) = a \cdot 1(t) \Leftrightarrow F(s) = 1/s$

Ramp signal: $f(t) = a \cdot t \Leftrightarrow F(s) = a/s^2$

Exp signal: $f(t) = e^{at} \Leftrightarrow F(s) = 1/(s-a)$

Sine signal: $f(t) = \sin(at) \Leftrightarrow F(s) = a/(s^2+a^2)$

LINEARITY:

$L(af(t) + bg(t)) = aLf(t) + bLg(t)$

DIFFERENTIATION

$L\frac{df(t)}{dt} = sF(s) - f(0_-)$

INTEGRATION

$L(\int_t f(s)ds) = F(s)/s$

Transfer Function

models a linear time-invariant system

$$G(s) = Y(s)/U(s) \Rightarrow Y(s) = G(s)U(s)$$

$$U(s) \longrightarrow G(s) \longrightarrow Y(s)$$

Poles and Zeros

$$F(s) = \frac{b_m s^m + b_{m-1}s^{m-1} + \dots + b_0}{a_n s^n + a_{n-1}s^{n-1} + \dots + a_0} = K\frac{\prod_{i=1}^{m}(s-z_i)}{\prod_{i=1}^{n}(s-p_i)} = \frac{c_1}{s-p_1} + \dots + \frac{c_n}{s-p_n}$$

$$\Rightarrow f(t) = \sum_{i=0}^{n} c_i e^{p_i t}$$

where p are the poles of the function and decide the system behavior

Transfer Function & Block Diagram

CPU model: $U(t) = \int_{s=0}^{t}(R_a(s) - R_c(s))ds \Leftrightarrow U(s) = \frac{R_a(s) - R_c(s)}{s} \Leftrightarrow G_0(s) = \frac{1}{s}$

let $K = C_s G_0(s)$

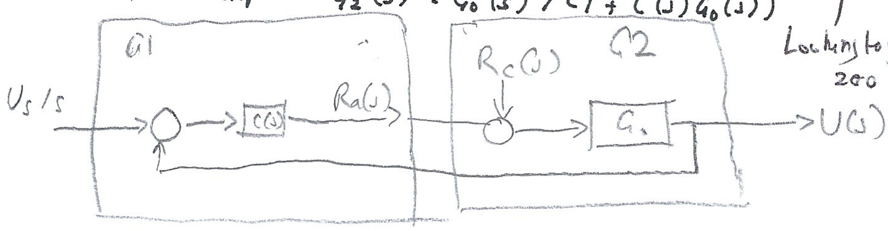
Pole $p_0 = -K < 0 \Leftrightarrow$ system is BIBO stable iff $K > 0$

Inputs : reference (set-point) $U_s(s) = U_s/s$; task completion rate $R_c(s)$

Close-loop system transfer functions have two possible inputs :

↳ $U_s(s)$ as input : $G_1(s) = C(s)G_0(s)/(1 + C(s)G_0(s))$

↳ $R_c(s)$ as input : $G_2(s) = G_0(s)/(1 + C(s)G_0(s))$



Looking to get value of $C(s) = ?$ to get zero steady-state error

$U(t) \rightarrow U(s)$

Output: $U(s) = G_1(s)U_s/s + G_2(s)R_c(s)$

(14)

BIBO STABILITY : bounded input results in bounded output. LTI system is BIBO stable if poles of transfer function in LHP ($\forall p_i, Re[p_i] < 0$)

Steady-State Error $e_{ss} = \lim_{t \to \infty} e(t) = \lim_{t \to \infty} (r(t) - y(t))$

($r(t)$: ref input, $y(t)$: system output)

↳ final value theorem:

$$\lim_{t \to \infty} f(t) = \lim_{s \to 0} sF(s)$$

$$e_{ss} = \lim_{t \to \infty} e(t) = \lim_{s \to 0} sE(s)$$

Robustness

① Disturbance Rejection: steady state error caused by external disturbances
↳ eg DOS

② Sensitivity: relative change in steady-state output / relative change of system parameters

Proportional - Integral - Derivative Control

↳ Proportional Control: $x(t) = Ke(t) \Leftrightarrow C(s) = k$ ↳ may have non-zero steady state error

↳ Integral Control: $x(t) = KK_i \int_0^t e(s)ds \Leftrightarrow C(s) = \frac{KK_i}{s}$ ↳ improves steady state tracking

↳ Derivative Control: $x(t) = KK_d \dot{e}(t) \Leftrightarrow C(s) = KK_d s$ ↳ may improve stability and transient response

PI Controller: stability

↳ $r_a(t) = K(e(t) + K_i \cdot \int_t e(t)dt)$; $C(s) = K(1 + K_i/s)$

Transfer Functions:

↳ U_s/s as input : $G_1(s) = (Ks + KK_i)/(s^2 + Ks + KK_i)$

↳ $R_c(s)$ as input : $G_2(s) = s/(s^2 + Ks + KK_i)$

Stability : poles $Re[P_0] < 0 \Leftrightarrow$ system is BIBO stable iff $K > 0$ & $K_i > 0$

↳ Steady-State Error

Completion rate $R_c(s) = R_c/s$

$$\text{System Response} = U(s) = \frac{U_s G_1(s)}{s} + \frac{R_c G_2(s)}{s} = \frac{(KU_s + R_c)s + KK_iU_s}{s(s^2 + Ks + KK_i)}$$

$e_{ss} = 0$ hence PI control accurately achieves desired response

Discrete Control & Modelling

z-transform : $f(k) \to F(z)$

$$F(z) = Z[f(k)] = \sum_{k=0}^{\infty} f(k) z^{-k}$$

Output in m^{th} sampling window $= V(m) = a_1 V(m-1) + a_2 V(m-2) + b_1 U(m-1) + b_2(m-2)$

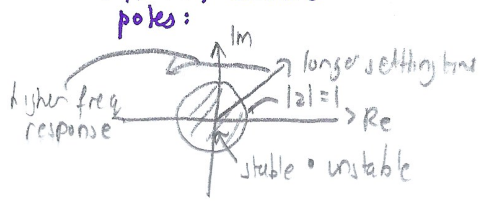
$U(m)$ is input in mth sampling window

$$V(z) = a_1 z^{-1} V(z) + a_2 z^{-2} V(z) + b_1 z^{-1} U(z) + b_2 z^{-2} U(z)$$

Transfer function $G(z) = (b_1 z + b_2)/(z^2 - a_1 z - a_2)$

Root Locus Analysis (1) stability boundary : $|z| = 1$, (2) settling time = dist from origin, (3) speed = location relative to Im axis

↳ Effect of discrete poles:



Advanced Control

① Robust noise → tolerance to noise
② Adaptive control
③ MIMO control
④ Stochastic control - minimise variance
⑤ Optimal controls - minimize cost function of control energy and error
⑥ Non-Linear Systems

(15)

Issues

1. Systems are non-linear
2. First-principles modelling is difficult
3. Tough to map control objectives to feedback control loops
4. Deeply embedded networking → decentralized

Router Queuing Behaviour in packet switched networks
↳ traditionally scheduling is very simple - better to increase speed of link and router.

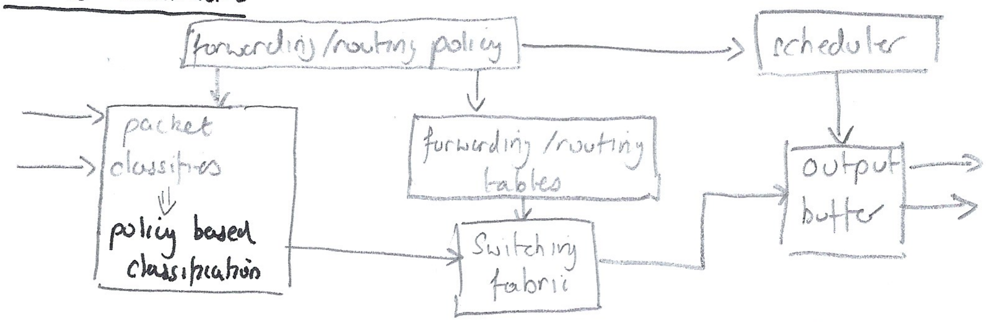
↳ ① DATA TRANSFER : individual packets, no recognition of flows and no signalling
↳ ② FORWARDING : based on per-datagram using forwarding tables
↳ no priority ~~stea~~ system
↳ ③ TRAFFIC PATTERNS ↳ add priority system to support QoS
↳ deal with congestion by tail drop where packets are lost when you put onto output queue
↳ can use FIFO queue not FCFS

SCHEDULING

requirements:
- ① Ease of implementation
- ② Fairness and protection → local/global, protect flows from another MAX-MIN
- ③ Performance bounds → per flow bounds, deterministic, probabilistic, data rate, delay, jitter loss
- ④ Admission control

Scheduler ① decides service order based on policy and (2) manages service (output) queues.

Router Schematic



Scheduler decides which output queue is serviced next

FCFS Scheduling : packets queued to outputs in order they arrive with no differentiation and no notions of flows

work-conserving scheduler → not idle if packets waiting

Conservation Law

Capacity is sum of utilisation and delay: $C = \sum_{n=1}^{N} \rho_n q_n$

↳ $\rho_n = \lambda_n \mu_n$
- λ_n → mean packet rate
- μ_n → mean per packet service rate
- ↳ mean link utilisation

Tradeoff between throughput and latency for different flows

Non-work-conserving schedules - receiver doesn't buffer receiving things

↳ Allows smoothing of packet flows (~~not~~ can be idle even if waiting packets)
- ↳ wait until packet is eligible for transmission
- ↳ higher end-to-end delay ↳ fixed time per router or fixed time across network
- ↳ Complex in practise
- ↳ Less jitter
- ↳ Downstream traffic more predictable
- ↳ Less buffer space

(16) Max-Min Fair Share Criteria: flows allocated resource in order of increasing demand
↳ they get no more than they need
↳ weighted max-min fair share possible. If fair, it provides protection

m_n = actual resource allocation to flow n
$= \min(x_n, M_n)$
↳ resource demand by flow n
↳ resource available to flow n

Capacity of resource (max resource) $= M_n = \dfrac{C - \sum_{i=1}^{n-1} m_i}{N-n+1}$

Dimensions
1. Priority Levels
2. Work conserving or not
3. Degree of aggregation
   ↳ flow granularity
   ↳ per application flow
   ↳ per user?
   ↳ per end-system?
   ↳ cost vs control
4. Servicing within a queue

Simple Priority Queuing: higher priority queues are serviced first
↳ not max-min fair - starvation

~~Generalised~~ Generalised Process Sharing
Work conserving with max-min fairness → can provide weighted max-min fair share.
NOT IMPLEMENTABLE ↳ round robin with infinitessimally low quantum
↳ used as a comparison (others emulate GPS).

↳ protection ⇒ (n-1) priority max

$\phi(n)$ - weight given to flow n, $1 \le n \le N$
$S(i,s,t)$, $1 \le i \le N$
↳ service to flow i in interval [s, t], flow i has non empty queue
$\dfrac{S(i,s,t)}{S(j,s,t)} \ge \dfrac{\phi(i)}{\phi(j)}$

$RFB = \left| \dfrac{S(i,s,t)}{g(i)} - \dfrac{S(j,s,t)}{g(j)} \right|$ ⇒ ↳ 1. Relative Fairness Bound - fairness of scheduler with respect to other flows it is servicing

$AFB = \left| \dfrac{S(i,s,t)}{g(i)} - \dfrac{G(i,s,t)}{g(i)} \right|$ ⇒ ↳ 2. Absolute Fairness Bound - fairness of scheduler compared to GPS for same flow

↳ GPS service for flow i in [s, t]

$g(i) = \min\{g(i,1), \dots, g(i,K)\}$

$g(i,k) = \dfrac{\phi(i,k)\, r(k)}{\sum_{j=1}^{N} \phi(j,k)}$ — $r(k)$: service rate of router k

i = flow number
k = router number

Weighted Round-Robin: Queues visited round robin in proportion to weights assigned
↳ different mean packet sizes → this is unpredictable and may cause unfairness
↳ service is fair over long timescales

If we instead compute packet size on the fly, we have Deficit Round-Robin
↳ each queue has deficit counter initially at zero
Scheduler attempts to serve one quantum of data from non-empty queue
↳ packet at head served if size ≤ quantum + dc
↳ else: dc += quantum
↳ set to max expected packet size

$RFB = 3T/r$
T: max packet service time
r: link rate

17

Weighted Fair Queing

calculate deadlines for each round for each packet size → calculated per flow. Need data store per flow and need info of destination. Hence, lookup necessary every time

Buffer Drop Policy: at full queue, drop in order of decreasing finish time.

↳ GPS emulation to get finish-numbers for packets in queue
- ↳ serves packets bit-by-bit round-robin
- ↳ time packet would have completed service under (bit by bit) GPS. Tags finish-number for each packet. Smallest finish number served first

↳ ROUND NUMBER → NB needs to be computed every time a packet arrives or leaves
- ↳ Execution of round by bit-by-bit round-robin server. Finish number calculated from round number
  - ↳ if queue empty: $fN = n(\text{bits in packet}) + \text{round\_number}$
  - ↳ else: $fN = \max(fNs \text{ in queue}) + n(\text{bits in packet})$

$F(i,k,t)$ = Finish-number for packet k on flow i at time t

$P(i,k,t)$ = size of packet k on flow i arriving at time t

$R(t)$ = round-number at time t — depends on number of active flows and their weights

$\phi(i)$ = weight given to flow i

$$F(i,k,t) = \max\{F(i,k-1,t), R(t)\} + P(i,k,t)$$

$$F_\phi(i,k,t) = \max\{F_\phi(i,k-1,t), R(t)\} + \frac{1}{\phi(i)} P(i,k,t)$$

Class-Based Queing

Assign a capacity and priority to each node which can borrow spare capacity from a parent hence meaning fine-grained flows are possible.

Queue Managment

- ↳ ① Ensuring buffers are available; ie memory management
- ↳ ② Organising packets within a queue
- ↳ ③ Packet dropping when a queue is full
- ↳ ④ Congestion control
  - ↳ ① Dealing with misbehaving sources
  - ↳ ② Source synchronisation
  - ↳ ③ Routing instability

↳ **Packet Dropping Policies**

1. Drop from tail
2. Drop from head
3. Random drop
4. Flush queue
5. Intelligent drop

End system reaction to Packet drops

1. TCP - works well
2. UDP - hurts real time adaptive flows — Can adapt for real time flows - use ECN or add multicast to get over issue of packet drop for TCP

Random Early Detection

Idea is to spot congestion before it happens so we can drop packets (can mark offending packets which are more likely to be dropped) implying preemptive congestion signal, stopping real congestion

$p(\text{packet drop}) \propto \text{queue length}$ — exp average: smooths reaction to short bursts

Datacenter Networks (QJump)

Use: ① commodity hardware ⇒ recent
② Static Network topology
③ Policies are under single admin domain
④ Cooperation
⑤ ~~Stat~~ Statistically multiplexed from internet

NB, no guarantees about latency
↳ Hadoop (one app) causes problems - queues caused by this slow down other applications

constraints
① unmodified apps
② Unmodified kernel code

Solve problems by applying queuing concepts in datacenters

Delays
↳ Queuing Delay D_q
↳ Servicing Delay D_s } causes queuing delay
↳ if we can bound servicing delay, rate limit hosts so we don't get queuing delay

$= n \times \frac{P}{R}$ (n = no. of sending hosts, P = packet size, R = edge speed)

So: → ① Network idle
② Hosts send ≤ P bytes
③ Wait (n × P/R) secs
④ Goto 1
} Network epoch

network epoch $= 2n \times \frac{P}{R}$

mesochronous compensation

Throughput $= \frac{R}{2n}$

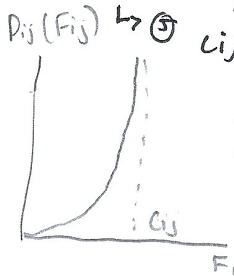
Can also add in priorities (→ hardware) then which allows for queue jumping

Optimization Based Routing

Framework:
↳ ① W set of source-destination pairs
↳ ② r_w : rate of sd pair w
↳ ③ P_w : set of paths between sd pair w
↳ ④ x_p : flow rate on path p.
↳ ⑤ c_{ij} : capacity of link i,j

⇒ Question is how to set rates on various paths
↳ they can by set ① centrally, ② at edges or ③ at routers

In routing problem, r_w given
In rate control problem, r_w variable



$F_{ij} = \sum x_p$
↳ all paths p crossing link i,j

ROUTING OPTIMIZATION PROBLEM ↳ minimizing either (i) system wide delay or maximize system wide utility
choose $\{x_p\}$

Aim: minimize $\sum_{\text{all links } ij} D_{ij}\left(F_{ij} = \sum_{\text{all links } ij} \frac{F_{ij}}{c_{ij} - F_{ij}}\right)$ subject to $\sum_{\text{all paths in } P_w} x_p = \forall w \in W$, ~~$x_p \geq 0$~~ all incoming flow routed to destination

At optimum: $\frac{\partial D(x^*)}{\partial x_{p}} = \frac{\partial D(x^*)}{\partial x_{p}}$ ∀ nonzero x_{p1}, x_{p2} in P_w ∀w

↳ Algo: ① ∀ sd pair w : evaluate $\frac{\partial D(x)}{\partial x_p} \cdots \frac{\partial D(x)}{\partial x_p}$ for all hw pairs for w
② Move small amount of flow to path with min marginal increase from other paths and repeat until all equal.

(19)

Can treat resource allocation as an optimization problem

Model

$L(s)$ = links used by source s
$U_s(x_s)$ = utility if source rate = x_s

USER PROBLEM: $\max \left(\frac{w_s}{p_s}\right) - w_s \quad w_s \geq 0$

System problem: $\underset{x_s \geq 0}{\text{maximise}} \sum_s U_s(x_s)$ subject to $\sum_{s \in S(l)} x_s \leq c_l \quad \forall l \in L$ (bandwidth less than capacity)

↳ $x_s = \frac{w_s}{p_s}$ ⇒ unit time; p_s ⇒ charge per unit flow for source ↳ COST

↳ network wants to maximise: $\sum_s w_s \log x_s$ (NETWORK PROBLEM)
↳ converges to relaxation of network problem

Idea is ∃ prices p_s, x_s, w_s, $w_s = p_s x_s$ st
$\{w_s\}$ solves user problem
$\{x_s\}$ solves network problem
$\{x_s\}$ solves system problem

x_s is fair if feasible and for $\{x_s^*\}$
↳ any other feasible:

$$\sum_{s \in S} \frac{x_s^* - x_s}{x_s} \leq 0$$

System properties

1. Convergence
2. Achieve objective
3. Benchmarks → ① $\max \sum_i U_i(x_i)$ st $Rx \leq c$
   ② Var x, R
4. Utility gap between joint system and benchmark

System Design

Putting together network resources (→ computation, storage and transmission resources.) to extract most usage
↳ in any system, some resources are more freely available than others

Some combination of: ① time, ② space, ③ computation, ④ money, ⑤ labour

Aim: To maximise performance metrics given resource constraints

Metrics

1. Time: response time, throughput, (degree of parallelism = response time × throughput)
2. Space: limit of available space (kB) and bandwidth (kb/s)
3. Computation: processing / unit time
4. Money: £

Social constraints

1. Standards
2. Market Requirements

Scaling is a design constraint (hard to measure) but very necessary for success. Also important for economies of scale.

Bottleneck: most constrained element in a system.

↳ removing bottleneck improves performance - but creates another bottleneck.

↳ Aim is generate balanced system where all resources are simultaneously bottlenecked

Techniques to trade off one resource for another

↳ ① Multiplexing - trades time and space for money

↳ increases response time and less space but costs less → economies of scale

↳ examples: ① Multiplexed link

② Shared memory

↳ server controls access to shared resource using schedule to resolve contention

Statistical Multiplexing

Resource has capacity C, shared by N identical tasks, each task requires capacity c. If $Nc \leq C$ then resource underloaded.

↳ if at most 10% tasks active $C \geq Nc/10$ enough

↳ statistical multiplexing gain

↳ Spatial multiplexing: expect only a fraction of tasks to be simultaneously active.

↳ Temporal multiplexing: expect task to be active only part of the time

② Pipelining

Break up task into independent subtasks - optimal if all subtasks take the same amount of time. But, ∃ dependencies. Pipeline is a special case of serially dependent subtasks

↳ best decomposition - degree of parallelism $= \frac{R}{S}$, maximized when $N = \frac{R}{S}$.

hence $S = R/N$

③ Batching

Group tasks together to amortize overhead - only works when overhead for N tasks < N time overhead for one task. Time taken to accumulate batch shouldn't be too long.

↳ tradeoff of reduced overhead for longer worst case response time and increased throughput

↳ Can also exploit locality: spatial and temporal through ④ caching

↳ 80/20 rule: 80% of time spent in 20% of code

↳ this is part that we should optimize

↳ Measure using Amdahl's Law

Execution time after improvement = $\frac{\text{(execution affected by improvement)}}{\text{amount of improvement}}$ + execution unaffected.

⑤ Hierarchy : decomposition of system into smaller pieces that depend only on parent ~~for~~ for proper execution
- ↳ No single point of control
- ↳ Higly scaleable
- ↳ But, leaf to leaf communication can be expensive

⑥ Binding and indirection
- ↳ translation from an abstraction to an instance
- ↳ Indirection : can bind automatically if translation table is stored in a well known place

⑦ Virtualization : combination of indirection and multiplexing
- ↳ refer to virtual instance that gets matched to instance at run time.
- ↳ can cleanly and dynamically reconfigure - build as if real resource is available

⑧ Randomization
- ↳ allows us to break a tie fairly

Soft State :
- ↳ memory in system that influences future behaviour.
- ↳ delete the state on a timer - refresh if you want to keep it.
- ↳ Automatically cleans up after a failure but increases bandwidth requirement
- ↳ Important to use explicit state exchange where network elements need to exchange state.

⑨ Hysteresis : when need to detect if value above or below threshold where the variable fluctuates near the threshold, we can use state-dependent threshold (hysteresis)

Data vs Control
- ↳ divide actions that happen once per data transfer and once per packet. - increase throughput by minimizing actions in data path. (→ data path vs control path)
- ↳ But keeping control information in data element has it's advantages
  - ↳ per packet QoS

⑩ Extensibility : good idea to leave hooks to allow for future growth

Tuning Existing Systems

① Measure
② Characterise workload
③ Build system model
④ Analyse
⑤ Implement