Advanced Computer Networks

Professor Ibrahim Matta

These notes build on a basic knowledge of computer networks, in particular the basic workings of TCP/IP protocols of the Internet. We cover mathematical and algorithmic foundations, and architectural design principles, of the essential networking functions of data transfer, transfer control, and management. Essential concepts of some techniques, such as optimization, control theory, queuing theory, and their application to networks are covered. Architectural considerations for building networks that are resilient, scalable, secure, and market-driven, are discussed.

These notes contain some problem sets to reinforce basic concepts.

Preface

Computer networks have become pervasive:

- On the rooftops of buildings in some neighborhood connecting routers in a multi-hop wireless fashion to provide access to local resources in that neighborhood as well as to remote resources over a shared Internet connection (MIT Roofnet project is an example),
- In the homes to allow users to access multimedia content stored locally or remotely,
- On planets to connect earth to Mars and other planets with each other,
- In the oceans connecting sensors on top of whales to collect measurement data on life in the oceans,
- And in many other settings like socially driven networks, e.g., Facebook, and file-sharing networks, e.g., bitTorrent.

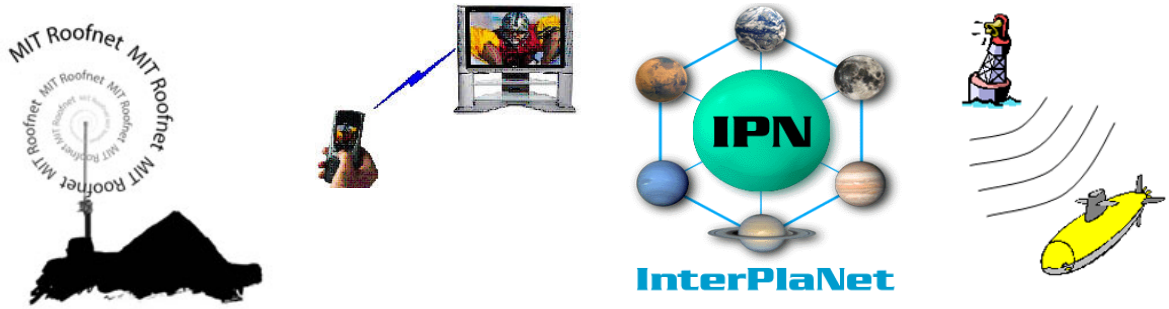


Figure 1: Networks are everywhere!

We want to become experts in the fundamentals of building distributed computing systems and networks, no matter where they arise, though we will use the Internet as the main case study. Besides fundamentals, we will get exposed to recent efforts and research questions that are yet to be fully answered.

We will discuss the fundamental functions needed for networking:

- **Data Transfer** concerns packetization (fragmentation/reassembly of Application Data Units into/from packets), sending/receiving packets, multiplexing, and queuing/scheduling. That happens at a short timescale, packet-level.
- **Data Transfer Control** concerns controlling the transfer of a stream of related packets (aka flow or connection), for example flow/congestion control, and error control. This operates over a medium timescale.
- **Management** usually happens at a slower (longer) timescale and often before the flow begins its transfer, e.g. locating the destination process, routing, access control, syncing the sender and receiver (aka connection management/handshaking).

We will discuss what the architecture that glues all these functions should look like.

What is "architecture"? It is a style of construction: what are the objects (processes and tasks) and how they relate to each other (i.e. interfaces).

Efforts to overcome limitations of the basic Internet architecture include overlays.

What is an "overlay"? We note that there must be an "underlay" then! An overlay is a set of processes communicating over a "virtual" network, that is, a network of virtual links. A virtual link creates the illusion of a direct point-to-point connection between communicating processes and could be implemented using Internet transport connections, for example, TCP resulting in a reliable link (no loss, no duplication, no reordering). Thus, the Internet TCP/IP forms the underlay. Notice that the Internet itself, in particular its Internet Protocol (IP), was built as an overlay over telephone lines – in this case, the underlay was the telephone network.

An overlay is just a network that we need to manage. We could re-use the TCP/IP protocols, or we could use a totally different (clean-slate) protocol suite.

We will also get exposed to the modeling of protocols so we can study their correctness and performance.

What is a model? A model is an abstraction that captures the essential elements of the system we want to model, but ignores unnecessary details.

For modeling and analysis, we will make use of techniques, such as queuing theory[1], optimization, and control theory[2], at some basic level.

[1] Note that queuing theory, like any other theory, has its limitations. Network calculus overcomes its limitation of assuming that arrival processes are Poisson, which is not the case in practice. Nevertheless, queuing theory provides important insights.
[2] Control theory overcomes limitations of game theory: classical game theoretic models capture synchronized or sequential rounds but does not model different timing / delays that are important in networking settings, and assume that behavior is rational and known

Thus, these notes assume basic knowledge of networking; basic probability, statistics, and queuing theory; and basic algorithms and data structures. Nevertheless, we will recap basic results and sometimes derive them from scratch to provide needed intuition and to provide a quick refresher and make the text as self-contained as possible.

Beyond Basic TCP/IP

The basic TCP/IP "best-effort" architecture has evolved and been extended over the years in support of performance-sensitive protocols and applications. This is exemplified by features implemented in **Cisco IOS** (Internet Operating Systems) and host operating systems like Linux[3], overlay architectures, convergent wireline-wireless architectures, and private (enterprise) networks. Specific examples include:

- **Integrated Services (IntServ) and RSVP**: the IntServ standard extends the best-effort service architecture of the original Internet with performance guaranteed service provided to individual flows. The basic building block is a per-queue/flow scheduling algorithm that allocated a minimum rate to each queue/flow. RSVP (ReSerVation Protocol) provides explicit signaling to set up buffer/rate for a flow along its path to the destination.
- **Differentiated Services (DiffServ)**: To reduce state overhead in routers, the DiffServ standard advocates per-class resource allocation, where a class is an aggregate of similar traffic flows. The tradeoff is "softer" performance guarantees due to potential interference between flows of the same class.
- **Multi Protocol Label Switching (MPLS)**: This standard provides the capability of establishing explicit paths under IP for the purposes of fast routing (at the MPLS level) akin to virtual circuits, traffic engineering (where IP traffic can be directed over alternate MPLS paths), and setting up a VPN (Virtual Private Network).
- **Traffic Engineering (or QoS/CoS Routing)**: MPLS paths can be allocated resources (e.g., buffer, bandwidth) along the links to serve the requirements (e.g., loss, delay) of an individual flow – known as Quality-of-Service (QoS) routing – or of a class of flows - known as Class-of-Service (CoS) routing.
- **Content Distribution Networks (CDN)**: CDNs, e.g., Akamai, have emerged that replicates content closer to the client so as to reduce response time.
- **Peer-to-Peer (P2P) Networks**: P2P networks, e.g., BitTorrent, have emerged to replace the traditional client-server communication model, wherein a peer can retrieve content from any other peer who had the content previously retrieved and stored.
- **Clean-slate Architectures**: More recently, the networking community has been investigating replacements of the TCP/IP architecture.

---

when in practice some players may be irrational or have unknown utility, e.g. exogenous wireless losses or attackers whose utility is typically to cause some damage to the network and its users.

[3] Linux can be configured on a machine to act as either a router or host.

Our focus in these notes is not on the specific implementations of these extensions but rather on the fundamental concepts, their mathematical modeling and algorithmic aspects. Learning the fundamentals will allow us to apply basic concepts and algorithms in different contexts, for example a given scheduling or routing algorithm could be employed at different levels of the network architecture such as at the network level inside a router for routing or scheduling packets and at the application/higher level inside a web server for scheduling web requests or for routing skype voice calls across the skype overlay network.

As we noted earlier, our focus will be on the design and *dynamics* of networks (and distributed computing systems in general). Dynamics often arise because of adaptations by both the users (or protocol entities on their behalf) in response to network performance they observe, and the network itself reacting to user demand in its allocation of resources (buffers, bandwidth, etc.)

To study such system dynamics, we will develop and study so-called *dynamic models*, wherein system variables (and sometimes parameters) change over time. Our goal will be to analyze their transient / dynamic / time-dependent performance and stability conditions. This is in contrast to *deterministic models* or *stochastic / queuing models* whose variables and parameters are constants or follow certain stationary probability distributions, respectively, and where we primarily analyze their steady-state performance. We will develop and study queuing models as well.

The following figure shows a block diagram that captures a typical adaptive (dynamical) system. We will revisit such diagrams later when studying different adaptive protocols employed by hosts and the network. In this figure, each block (box) represents a relationship between its input and output. "Users" block represents a protocol like TCP adapting its sending rate based on observed/inferred packet losses, represented here abstractly as "prices". The "demand" from the users is routed by the network, represented by the "Plant" block, to cause certain "load" on each "resource" (e.g., an output link/interface of a router). Whenever the resource exceeds a certain load level, it starts dropping packets – more abstractly, a "congestion signal" or "price" gets generated that users observe after some amount of propagation delay, known as "feedback delay". In response to prices, users adjust their demand and the cycle repeats over time until the system may or may not converge to a stable point (state) when supply matches demand. We generically call such system a *feedback control system*. It is worth noting that this feedback control system is typical of many other real-life systems, e.g., the time-varying demand and supply for oil as a resource, wherein oil companies set prices based on demand. In addition to prices driven by demand, we call them "load-dependent prices", there might also be external, load-independent, prices, we call "exogenous prices" – in a networking setting, these might be wireless losses due to transmission errors or packet losses due to a denial-of-service attack! We will study the convergence and stability of dynamical systems using control theory.

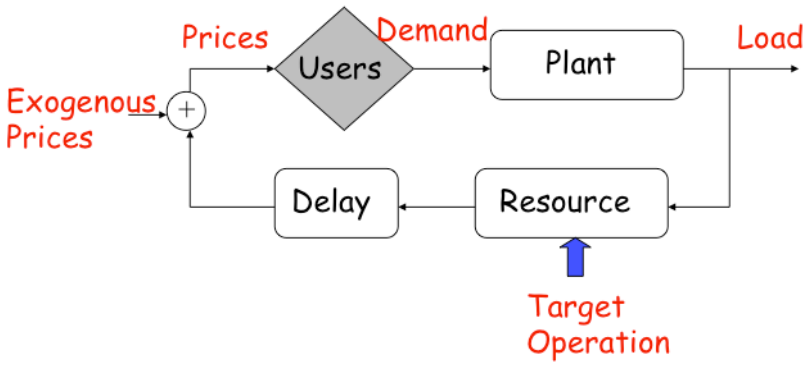


Figure 2: A Feedback Control System.

The TCP/IP Architecture Revisited

We revisit here the TCP/IP (Internet) architecture, starting from its ARPANET precursor, how evolving requirements over the years have put the original design under stress, and how the original design has been extended to respond to those new requirements.

The ARPANET (Advanced Research Projects Agency Network) was the first wide-area network developed in the US. It consisted of switches connected by long-distance links. The original goal was to develop a robust communication system that can survive nuclear attacks [Paul Baran, 1960-64]:

> *"Both the US and USSR were building hair-trigger nuclear ballistic missile systems …long-distance communication networks at that time were extremely vulnerable …That was the issue. Here a most dangerous situation was created by the lack of a survivable communication system."* (Baran in Abbate, 10)

If the network is not "robust", then if it gets attacked by one side and goes down, the other side won't be able to communicate and control its missile system to retaliate. A dangerous situation arises because there will be no deterrence for the other side from initiating an attack!

Paul Baran developed a design for the ARPANET that had the following elements:

- **Packet switching technology**: this allows statistical multiplexing of packets, which means bandwidth of flows with no packets to send gets used up by other active flows. This reduces the amount of total bandwidth allocated.
- **Totally distributed**: all nodes (switches) are equal, i.e. no master-slave relationships. This is considered a first instantiation of the so-called peer-to-peer (P2P) communication model, a concept that became popular in the late 1990's at the application level as an alternative to the traditional client-server model for serving content.
- **Robust**: this is accomplished by adequate physical redundancy, adaptive routing that directs traffic to less loaded or better performing paths, and priority

forwarding to transit over new packets. We observe that the latter is a form of network flow control and differentiated service!

- **End-to-end error control**: the ends (i.e. producers and consumers of the data) tolerate and recover from errors. The switches themselves do not retransmit lost packets. In other words, complexity of error control is pushed to the edges of the network (the ends or hosts) while keeping the network core (switches) simple.

The design of the original Internet had many features in common. The primary (original) requirements, from high to low priority, were:

- **Multiplexing**: to achieve cost-effective / efficient use of bandwidth, packet switching, statistical multiplexing was adopted.[4]
- **Survivability**: to achieve robustness, error control is employed at the ends (so-called "end-to-end principle"), the network is *stateless*, and packets are delivered as *datagrams*. By stateless, we mean that the state of a conversation (flow) is maintained at the end hosts, not within switches/routers. Datagrams are delivered independently, i.e. packets from the same flow may take different paths to their destination, perhaps after a failure on the primary path. [5]
- **Service generality**: Originally transport (TCP for reliable service) and routing (IP) were considered one protocol. By separating TCP and IP, it became possible for applications to use transport services provided by protocols other than TCP, namely UDP (for unreliable service). Over the years, other transport protocols were developed, e.g. RTP (for real-time transport).
- **Diverse network technologies**: IP adopted a "best-effort" delivery service, i.e. IP assumed a least common denominator service from underlying networks it interconnects, where no reliability or performance quality is expected. Nor does IP leverage any capabilities that each individual network may have. In other words, the Internet has become one large best-effort network that the ends have to live with and adapt to! That proved to be really hard, as we will see. [6]

The Internet: End-to-End Principles

The end-to-end principle mentioned above is claimed by some to be the cornerstone of the Internet's original design, and was the impetus for its phenomenal success. This is how its designers define it:

---

[4] It's worth noting that the use of packets by itself was no choice really, given that in a data network, what we want to send is a buffer of bits, i.e. packet!

[5] Some would argue that the concept of datagrams was the main design choice that made the Internet a success. However, this concept has been earlier employed in the French CYCLADES network, so credit is due there!

[6] This many-transports over IP, and IP over many-networks gave rise to the so-called "hour glass" model (after an hour glass shape). One may argue that the best-effort IP routing just glued constituent networks into one single giant best-effort network, far from a real internetwork of diverse networks, each with its own identity and capabilities!

A function that can be entirely accomplished in an end node is left to that node, and the communication state is kept only in that node

For example, since applications requiring reliability are ultimately the ones to ensure data integrity as data finally gets written on the disk at the receiver, the end hosts should be where error control is performed, and not inside the network within routers. This is exemplified by TCP running only at hosts. That leads to the idea of "fate sharing": the end host dies, the connection (flow) state gets lost; routers die, connection state is unaffected and the hosts (TCP) can retransmit lost packets resuming reliable delivery.

Another (related) definition of the end-to-end principle is:

The network is built with no knowledge of, or support for, any specific application or class of applications

That is consistent with a network that makes minimal assumptions about the applications using it. This is also in line with Occam's razor, a philosophy of succinctness articulated by a 14th century English logician wherein unnecessary details are *shaved off* from a model: "the simplest of competing theories/models is preferred to the more complex" [Merriam-Webster] The Internet complied with this philosophy by keeping the core of the network simple (i.e., stateless). [7] We observe that if the network does not provide sufficient support to applications, then application processes may end up doing networking themselves to overcome the network's limitations. This is exemplified by overlays (e.g. skype) and CDNs (e.g. Akamai). It's easy to imagine chaos could then ensue when such overlays are unaware of the state of the underlay! Recall Albert Einstein's quote: "Make everything as simple as possible, but not simpler." An important question is then: *was the end-to-end principle good for the Internet or impediment to a cleaner architecture?* We will discuss this question in later sections.

The Internet: Secondary / Later Requirements

The design of the Internet also had secondary requirements and those that emerged over the years:

- **Distributed management**: this can be seen from the Internet's two-tiered routing. Routing is based on the notion of Autonomous System (AS) as an independently managed network/domain. Domains exchange routing information using an EGP protocol (Exterior Gateway Protocol); while inside each domain an IGP (Interior Gateway Protocol) protocol is run. IGP could use a link-state approach, like OSPF and IS-IS, where each router collects a full view of the topology; or IGP could use

[7] The RISC (Reduced Instruction Set Computer) architecture is an non-networking example of a simple computer architecture that arguably follows Occam's razor.

a distance-vector approach, like RIP, where routers exchange their best distances to destinations. BGP is the de-facto EGP in the Internet. BGP uses a path-vector approach that extends distances with path information.

- **Security**: To respond to new requirements for security, the Internet architecture has been extended with a "shim" layer that sits between the application and the transport layer to encrypt application data and provide secure channels. The de-facto shim protocol is TLS (Transport Layer Security), whose predecessor is SSL (Secure Sockets Layer).
- **Mobility**: To respond to new requirements for supporting the mobility of nodes, the mobile-IP standard was developed. Given that when a host moves and changes its point-of-attachment to the network, its IP address changes which causes the ID of the transport connection to change. As a result, transport connections break. We observe that coupling the connection ID with the location/address has been a major source of problems that had faced the Internet and challenged its original design. We will revisit naming and addressing in later sections.
- **Resource allocation**: To support new requirements for supporting better than just "best-effort" service, standards like IntServ, RSVP, DiffServ, and MPLS, have been developed. This enhanced network support was motivated by real-time applications like streaming voice and video. Metrics such as fairness (i.e. equal resource allocation among flows) and QoS (aka controlled unfairness) have been considered.

Other (new) requirements include:

- **Accountability**: This gave rise to value-based pricing. For example, tools that measure throughput promised by Internet Service Providers (ISP) were developed. [8]
- **Trust**: The commercialization of the Internet meant that users could no longer trust each other. Devices like firewalls and traffic filters were placed in front of private networks to control access from the outside, so "unauthorized" packets get dropped.
- **Less sophisticated users**: To support users with relatively more limited resources (aka thin clients), proxies were placed in front of them to pre-process complex data on their behalf, e.g. sending them only text-only version of web pages.

In fact, in-line devices like firewalls, traffic filters/shapers, proxies, web caches, etc. have collectively been called "middleboxes". They often perform some application-specific processing by inspecting arriving packets beyond their IP header, e.g. looking at transport protocol port numbers. This has been called Deep Packet Inspection

[8] See www.speakeasy.net for a tool to measure/test your computer's download and upload speeds in support of voice-over-IP (VoIP) applications.

(DPI). Some consider such DPI as breaking the end-to-end principle or destroying the pure original design of the Internet. Does it? Looking deeper into the end-to-end arguments, it is only a design guideline. Moreover, violating its purest form is acceptable for the purpose of achieving low-cost performance enhancements as long as end-to-end semantics (correctness) of applications is not violated. For example, a web cache that always returns an up-to-date web page/object does not violate the end-to-end principle. However, if the web server inserts dynamic advertisements that the web cache ignores, then we could argue that the end-to-end principle is violated. How about a firewall? Well, one end puts up the firewall and knows about it, so one could argue that the firewall as a proxy to that end does not violate the end-to-end principle. In summary, as long as the ends are aware of the middleboxes and what they are doing, one could argue that the end-to-end principle is not contradicted. We will revisit the relevance of the end-to-end principle in later sections.

A Different Approach to Reliability

We have discussed above how enhancements to the basic datagram delivery service have been developed in response to changing requirements.[9] Another approach to build a robust network is to make use of *circuits* as in a telephone network.[10]

In a telephone system, system reliability is achieved mostly by ensuring every component is reliable, i.e. it has minimal downtime. Furthermore, the telephone network is tightly controlled, where, unlike the Internet, there is a separate control plane for signaling and access control so circuits get pre-allocated and hence service predictability in the data plane can be achieved, i.e. a voice flow is guaranteed 64Kbps with no loss.[11]

Thus, in a phone network, end-specific state is maintained inside the network to keep track of time slots statically allocated to the voice flow and the next-hop for its fixed path, i.e. we say that the network is circuit-switched and stateful. Furthermore, the phone network maintains *hard state*, i.e. the flow state has to be removed using explicit removal messages. In contrast, in a *soft state* approach, the flow state is removed after a timeout period unless refreshed, i.e. timer-based state expiration. We will discuss the modeling and evaluation of such flow management approaches in later sections.

---

[9] Some would say mostly patches got deployed!

[10] In a telephone network, circuits are allocated statically using TDM (Time Division Multiplexing). The bandwidth allocated to a voice circuit with no traffic can not be used by another voice flow. This is in contrast to statistical multiplexing of packets.

[11] 4KHz (cycles/sec) is the highest frequency component of a voice signal. Nyquist says to digitize the signal by sampling it at twice that rate to be able to completely reconstruct the voice signal at the receiver. Assuming 256 quantization (amplitude/height) levels (i.e., 8 bits), then a voice flow would require 2*4K samples/sec * 8 bits/sample = 64K bits per second.

The use of circuits has found its place in some architectural proposals. One idea is to use circuits in the core when aggregate traffic is less bursty (so, bandwidth won't be wasted), and datagrams around the edges. Some studies have also found circuit switches simpler than software-intensive IP routers!

Telephone networks have also employed adaptive routing of voice calls. Specifically, on call setup, the network may use two-hop paths if one-hop path is full (i.e. it can not carry one more voice circuit).

The Network is there to Support Applications After All

Telephone networks were originally designed to carry voice and support their 64Kbps rate requirement. Cable networks were originally designed to carry video. On the other hand, the Internet was designed to carry data and accommodate their bursty behavior (e.g., a user downloads a web page and then typically sits idle, thinking, before initiating another transfer). As a form of convergence, networks have started to serve other kinds of applications that they were not originally designed to serve.

In general, we can categorize applications into:

- **Real-time**: e.g., voice, video, emergency control, stock quotes, ...
- **Non-real-time** (or best-effort): e.g., telnet, ftp, …

The Internet was originally designed for non-real-time applications. Real-time applications, like streaming voice or video, have timing requirements. Timing requirements can be classified into:

- **Hard with deterministic or guaranteed requirements**: e.g., no packet loss, packet delay must be less than a given deadline bound, the difference in delays of any two packets must be less than a jitter (delay variability) bound, … This kind of requirements give real-time applications a highly stable rate over the network, where packets (e.g. carrying voice samples or video frames) must be played back at a constant rate equal to their generation/sending rate. In this case, a packet is considered lost not only because of buffer overflow but also if its playback time (deadline) is missed. These deadlines are increasingly violated as delay jitter (variance) increases and packets are received after their deadline. One way to reduce deadline violations in the presence of delay jitter, which mostly arises because of queuing inside the network, is to push (delay) the playback point so packets are there in the receiving host's buffers when it's time to play them back. However, this delayed playback may reduce the interactivity of the application. Thus, a well-designed network with little jitter would require fewer buffers at the receiving host to absorb delay variation and the playback point can be set as soon as the propagation delay over the network allows. **[Insert figure]**
- **Soft with statistical or probabilistic requirements**: e.g., no more than x% of packets lost or experience delay greater than a given deadline. This latter requirement bounds the tail of the delay distribution of packets.

The next question is: is the original end-to-end design of the Internet enough to support such real-time applications? Is it enough to employ TCP for error control, oblivious to any delay requirements? We discuss these issues next.

Is End-to-End Control (in the style of TCP) Enough?

Here we answer the question:

> *Is the original end-to-end design of the Internet enough to support real-time applications? Is it enough to support their delay and jitter (delay variance) requirements? Is it enough to employ TCP for error control, oblivious to any delay requirements?*

Intuitively, with common FCFS schedulers at routers, delay and delay variance increase very rapidly with load. Furthermore, if the transport protocol is oblivious to delay requirements, it may attempt to retransmit a lost packet whose deadline has already expired – a futile exercise and waste of resources!

M/M/1 Queuing Model

In answering these questions more analytically, we will start introducing modeling techniques from queuing theory. We start with a very simple queuing model where we model the network as a single FCFS infinite queue with a Poisson arrival process and exponentially distributed service times. We denote such queuing system as M/M/1/∞/FCFS, where:

"M" stands for Markovian or memoryless, i.e. whenever an event occurs, the new state of the system depends only on its current state, and not its history. An example memoryless process is the Poisson process that gives the probability of the number of arrivals x in a time period t as:

$$P(X = x \text{ in period } t) = e^{-\lambda t} \frac{(\lambda t)^x}{x!}$$

where λ denotes the arrival rate. Another memoryless process is the exponential distribution that gives the service time t given an average service time of $1/\mu$ (i.e. departure rate is μ) as:

$$f(t) = \mu e^{-\mu t}$$

Note that to say that a process is Poisson is equivalent to saying that it generates exponentially distributed inter-event (arrival / departure) times. [**Exercise:** Show that this is the case.]

The rest of the M/M/1/∞/FCFS notation indicates one queue that is infinite and that employs a FCFS scheduling discipline. We also use M/M/1 for short.

Given an arrival rate λ and a service rate μ, an M/M/1 queue is stable iff λ < μ, i.e. ρ = λ/μ < 1. ρ is called the traffic intensity or load.

To analyze such a system, we first have to define its "state". Let us define the state *n* as the number of packets in the system (waiting in the queue + in service): n = 0, 1, 2, …

Then, we can show that the average (expected) number of packets in the system $E(n) = \frac{\rho}{1-\rho}$. This means that as ρ approaches 1, i.e., the system approaches saturation, the average number of packets in the system increases dramatically. The same can be said for average packet delay and variance.

To obtain the average (expected) packet delay, we can use the PASTA property (Poisson Arrivals See Time Averages): a new arrival sees the average state of the queue. Thus,

$$E(d) = \frac{1}{\mu}\,(E(n) + 1) = \frac{1}{\mu(1-\rho)}$$

The "+1" accounts for the service time of the new arriving packet itself. A typical objective of the system would be to limit the average packet delay *E(d)* to be less than a given delay bound *D*. This can be achieved by limiting ρ. Similarly for the variance, which is given by:

$$V(d) = \frac{1}{\mu^2\,(1-\rho)^2}$$

In some sense, TCP imposes this limit on ρ by doing congestion control. In reality, buffers are finite and as load increases, buffer overflows and router starts dropping packets. Once TCP detects packet loss, it backs off its transmission window. We analyze TCP throughput below.

[**Exercise:** Derive *E(n)* by developing a Markov Chain, i.e. birth-death model where two events don't happen at the same time. In steady state, equate the flow into a state to the flow out of that state. Note that ρ also denotes the utilization of the system.]

Little's Law:

A very useful formula for queuing systems is called Little's law. It relates *E(n), E(d),* and λ. At steady state, λ is also the system's throughput (actual packet departure rate):

$$E(n) = \lambda * E(d)$$

This result is very powerful as it applies to any (stable) system in steady state. This result is independent of the arrival and service distributions, and of the scheduling algorithm used.

TCP Throughput Analysis

Here we develop a simple model to derive the throughput, more precisely, the sending rate, of a TCP connection. We will derive the so-called "inverse-$\sqrt{p}$ formula" that shows that TCP throughput, λ, depends on:

- The transmission window dynamics $w(t)$, dominated by the Additive-Increase-Multiplicative-Decrease (AIMD) adaptation,
- The underlying path characteristics, assuming constant packet loss probability *p* and constant RTT (Round-Trip Time).

In later sections, we will see more detailed models of TCP. But, modeling only the AIMD operation of TCP:

Additive-increase rule (AI): $w(t + RTT) = w(t) + 1$, if no packet loss is detected.

Multiplicative-decrease (MD): $w(t + \delta t) = w(t)/2$, if packet loss is detected (observing duplicate acknowledgments).

Noting that $\lambda = {}^{w}/_{RTT}$, we can rewrite these two adaptation rules in terms of sending rate:

AI: $\lambda(t + RTT) = \lambda(t) + 1/RTT$

MD: $\lambda(t + \delta t) = \lambda(t)/2$.

Note that we say that TCP operates at the "cliff" of the throughput-load curve to *induce* packet loss – the "cliff" is the point where throughput (more precisely, "goodput" or output rate at which *new* data get delivered) starts to decrease as the input rate increases, a phenomenon known as "congestion collapse". In response to packet loss, TCP reduces load to bound delay but there's still oscillation in delay, upper bounded by the buffer size. **[Insert figure]**

Assuming a periodic packet loss model, we will show that the steady-state sending rate of TCP is given by:

$$\lambda = \frac{\sqrt{3/2}}{\sqrt{p} * RTT}$$

Define a *congestion epoch* as the time period *T* between two packet losses. During a congestion epoch, assuming at steady state the window oscillates between W and $W/2$, the number of packets sent N is given by:

$$\frac{W}{2} + \left(\frac{W}{2} + 1\right) + \left(\frac{W}{2} + 2\right) + \cdots + \left(\frac{W}{2} + \frac{W}{2}\right) \approx \frac{3}{8}W^2$$

Given that $p = \frac{1}{N}$, we obtain $W = 2\sqrt{\frac{2/3}{p}}$. Given that $\lambda = \frac{N}{T} = \frac{N}{\left(\frac{W}{2}\right) * RTT}$, we obtain the "inverse-$\sqrt{p}$" throughput equation.

We observe that the throughput of a TCP user is inversely proportional to its RTT. Thus, for two TCP users on different hosts sharing the same bottleneck (i.e. they observe the same packet loss probability *p*), a user with shorter RTT would obtain higher throughput than that of a user with longer RTT.

We can also arrive at the same conclusion by tracing the *dynamics* of adaptations of two competing users graphically in a two-dimensional space (λ_1, λ_2). The ideal operating point to which the trajectory of (λ_1, λ_2) should converge is where the system is both efficient (i.e. $\lambda_1 + \lambda_2 = C$) and fair (i.e. $\lambda_1 = \lambda_2$), that is the point where $\lambda_1 = \lambda_2 = C/2$. If the two users experience the same RTT, then each user converges to the same average throughput. On the other hand, if they experience different RTT, then the user with smaller RTT converges to a higher average throughput. **[Insert figure]**

This can be viewed as a form of *uncontrolled* unfairness in the sharing of network bandwidth. This leads to the observation that:

Users should not trust the network!

Furthermore, some users may not "play by the rules" (i.e., TCP rules) and reduce their sending rates upon congestion, *i.e.* they are not TCP-friendly sources like a streaming voice or video UDP-based application, given that the error control of TCP does not account for any timing/delay requirements. This leads to the observation that:

The Network should not trust the users!

This *uncontrolled sharing or allocation* of resources has been a source of problems for the Internet. For one, it led to the *tussle* between users/applications and Internet Service Providers. For example, ISPs have tended to install traffic filters to curb the appetite of some bandwidth-hungry applications like P2P applications (e.g., BitTorrent). And of course, some people cry "net neutrality"! Poor ISPs!!

We have discussed the consequences of the basic Internet architecture failing to provide *controlled* allocation of resources – the users couldn't trust the network, and the network couldn't trust the users. Even David Clark, who was involved with the Internet in its early days, admitted this erosion of trust [David Clark & Marjory Blumenthal, 2000]:

> *"The simple model of the early Internet – a group of mutually trusting users attached to a transparent network – is gone forever."*

"Making the network more trustworthy, while the end-points cannot be trusted, seems to imply more mechanism in the center of the network to enforce "good" behavior."

This admits that not all users are trustworthy (i.e. they do what they are supposed to do and "play by the rules") and so the network had to enforce correct, desirable behavior, e.g. using firewalls, traffic filters, etc.

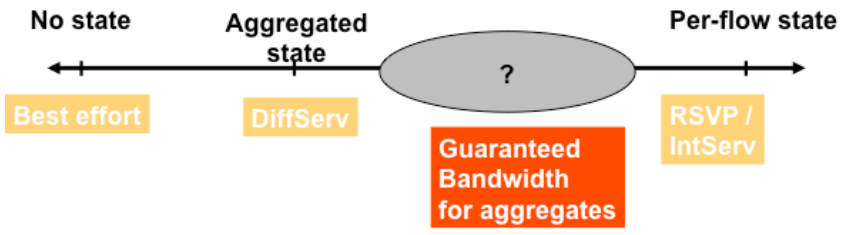
The deviation from the original "best-effort" "end-to-end" design of the Internet has also been justified on the basis of economic considerations.

"It is in the nature of private enterprise to separate users into different tiers with different benefits and price them accordingly."

"Low prices and ease of use are becoming more important than ever, suggesting growing appeal of bundled and managed offerings over do it yourself technology."

This admits that the network may provide *controlled* allocation of resources, also known as *quality of service* (QoS), rather than the ends attempting to overcome limited support from the network.

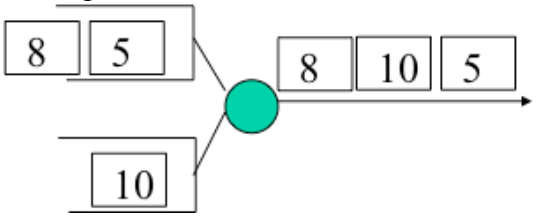
As we have mentioned earlier, this gave rise to efforts, such as IntServ and DiffServ, that tradeoff some complexity, namely, flow state inside the network, for more service quality (predictability). A balanced approach was traffic routing using virtual circuits coupled with non-FCFS guaranteed-service scheduling along the paths, so that it runs under IP (thus supporting many applications) and it "runs over anything" (i.e. over a diverse set of network technologies). The goal is to provide bandwidth-guaranteed fault-tolerant virtual circuits for typically few *aggregate* flows, e.g. going from one set of access networks to another as opposed to individual host-to-host flows. Such architectural enhancements in traffic engineering/routing and resource provisioning have been exemplified by standards, such as MPLS and Class-based Weighted Fair Queuing (CBWFQ), implemented by Cisco, Juniper, Linux, etc.



Weighted Fair Queuing (WFQ)

In FCFS scheduling, all packets from all flows share the same buffer, so they interfere with each other in the sense that a burst of packets from one flow can cause increase delay and jitter for packets of other flows. This increases worst-case delay and jitter.

A non-FCFS scheduler would have a separate queue for each flow, thus flows can be *isolated* from each other. Ideally, we can then allocate each queue (flow) the same equal service rate (capacity). This can be achieved by serving one bit from each queue in each round, known as "ideal bit-by-bit round-robin (RR)" scheduling, thus the worst-case waiting time for service can be bounded and delay *guarantees* can be provided. A weighted service allocation version can be obtained by serving an unequal number of bits from each queue in each round. In practice, we can only serve packets, not bits. A packetized version, known as Fair Queuing (or Weighted FQ), would approximate the ideal bit-by-bit RR. FQ stamps an arriving packet with its finishing time if ideal bit-by-bit RR *were* used, i.e. ideal RR is simulated. Then, FQ serves packets in the order of increasing finishing times. The following figure shows an example of FQ in action. The timestamps of the packet of size 5 bits and that of size 10 bits are 5 and 10, respectively, as they would have finished their transmission in these rounds under ideal bit-by-bit RR. The timestamp of the packet of size 8 is 13.



One can see that in practice, when packet transmission does not get preempted, FQ is only an approximation of ideal bit-by-bit RR. For example, if a *large* packet arrives to an empty system of two queues and joins the first queue, this large packet gets scheduled and transmitted. Immediately afterwards, if a second *small* packet that belongs to the second queue (flow) joins, although this small packet would have finished almost right away after its arrival, now without preemption, it has to wait for the large packet to finish its transmission. Thus, the approximation error in terms of the absolute difference between the packet delay under FQ and packet delay under the ideal bit-by-bit RR is upper-bounded by the maximum packet size divided by the link capacity.

The WFQ version can allocate different service rates μ_i to different queues (flows), i.e.

$$\mu_i \geq \frac{w_i}{\sum w_j} C$$

where w_i is the weight allocated to flow *i*, and *C* is the link capacity. The right-hand side of this inequality represents the minimum service rate guaranteed to the flow. This minimum service rate is received when all queues are active (i.e. non-empty), otherwise a non-empty queue receives higher service rate since no capacity is wasted due to statistical multiplexing and rate allocated to empty queues gets proportionally assigned to non-empty queues.

How should we assign weights in WFQ?

The capacity allocated to a queue (flow) depends on:

- Delay or jitter requirement of the flow
- Shape of the traffic generated by the source of the flow (i.e. its characteristics)

Assume the source generates traffic at the average rate of λ (packets/sec) and variance σ^2. Then, the service rate that needs to be *effectively* allocated to that flow, called *effective bandwidth* in the literature, is greater than or equal to λ, so as to accommodate the variance (burstiness) of the source traffic. We want the effective bandwidth (or, its sum over all sources) to be less than the total capacity of the link.

Discrete-Time Queuing Model

We next consider a simple single queue *discrete-time* model where time is slotted, and the number of packets arriving to the queue in slot *n* is a random variable A_n with average λ and variance σ^2. Assume that the service time of a packet is a constant of one time slot. Define the state of the system by the number of packets in the queue and in service, y_n in slot *n*. Our goal is to compute *E(y)*, and then find the effective bandwidth that satisfies $E(y) < D$, where *D* is a given delay bound.

We start with the difference equation that describes the evolution of y_n over time:

$$y_{n+1} = y_n - 1\{y_n > 0\} + A_n$$

Taking the expectation of both sides:

$$E(y_{n+1}) = E(y_n) - E(1\{y_n > 0\}) + E(A_n)$$

At steady state,

$$E(y) = E(y) - P(y > 0) + \lambda$$

Thus, $P(y > 0) = \lambda$. Recall that *E(x+y) = E(x) + E(y)*. Also, $E\big(f(x)\big) = \Sigma f(x)P(x)$, thus $E(1\{y > 0\}) = \sum_{y>0} 1.P(y) = P(y > 0)$, which is the probability that the queue is not empty.

Squaring both sides of the difference equation and taking the expectation, we obtain:

$$E(y^2) = E((y + A)^2) - E(2(y + A)\,1\{y > 0\}) + E(1\{y > 0\})^2$$

Recall that $E(xy) = E(x)E(y)$ if *x* and *y* are independent. We note that the arrival process and the number of queued packets are independent, i.e. intuitively observing one quantity is not sufficient to infer the other one.

$$E(y^2) = E(y^2) + E(A^2) + 2\,E(y)E(A) - 2E(y.1\{y > 0\}) - 2E(A).E(1\{y > 0\}) + E(1\{y > 0\})^2$$

Note that $E(y.1\{y > 0\}) = \sum y\,P(y) = E(y)$, $E(1\{y > 0\})^2 = P(y > 0) = \lambda$, and $\sigma^2 = E(A^2) - \lambda^2$. Simplifying, we get:

$$0 = \sigma^2 + \lambda^2 + 2\,\lambda E(y) - 2E(y) - 2\lambda^2 + \lambda$$

$$E(y) = \frac{\lambda(1 - \lambda) + \sigma^2}{2(1 - \lambda)}$$

Given the service time is one time slot, to bound waiting time by *D*, we want:

$$\frac{\lambda}{2} + \frac{\sigma^2}{2(1-\lambda)} < D$$

We know, for stability, that $\lambda < 1$. Thus, we can write:

$$\frac{\lambda}{2} + \frac{\sigma^2}{2(1-\lambda)} < \frac{1}{2} + \frac{\sigma^2}{2(1-\lambda)} < D$$

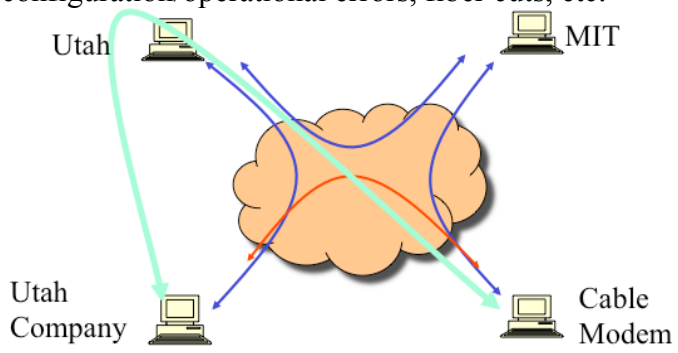
Rearranging, we get:

$$\lambda + \frac{\sigma^2}{(2D-1)} < 1$$

We observe that the right-hand side of this inequality represents the unit service rate. Also the left-hand side represents the effective bandwidth needed to allocate that source to meet the delay bound, which is equal to the average arrival rate, plus a quantity that depends on the traffic variance and the desired delay bound. In particular, a higher traffic variance or tighter (i.e. smaller) delay bound increases the effective bandwidth required to be allocated to that source.

Overlays over IP or Underlays

Another kind of attempts to overcome the limitations of the basic "best-effort" Internet architecture had been the building of "overlays". An overlay is a virtual network between processes linked (communicating) with (TCP or UDP) transport connections. These virtual links may physically run over an underlay of performance-guaranteed (e.g. MPLS) paths. Thus, traffic may be re-routed through one or more intermediate hosts to avoid the possibly congested or unavailable direct path. Congestion may arise due to traffic overload, denial-of-service attacks, etc. Path outages may be because of configuration/operational errors, fiber cuts, etc.

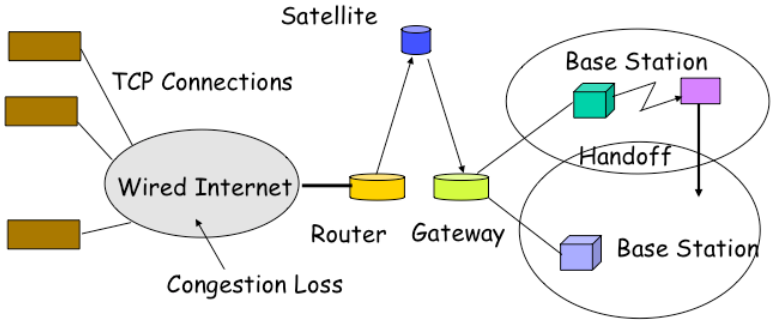


Observing that the source and destination overlay processes (nodes) can communicate through intermediate overlay processes (nodes) over reliable TCP connections, one question we may ask: do overlays "violate" the end-to-end principle? Do they break the end-to-end semantics of TCP when data packets received at an intermediate node get acknowledged before the final destination node actually receives the data? Well, overlay processes only perform routing over virtual links that happen to be loss-free. But, as long as the source and destination nodes (the ends) still maintain a higher-level TCP or TCP-like connection to provide service to their application processes, then the end-to-end semantics are not violated. As we will discuss later, there seems to be an inherently recursive structure that emerges where overlay networks, independently managed, are

built on top of virtual or physical links, which themselves constitute communication over underlying networks that are also independently managed. We will discuss our recursive internet architecture that generalizes and formalizes this organization.

Challenged Internet

The basic Internet architecture has been further challenged with its extensions with wireless links, including cellular radio, satellite and acoustic. One problem is that the widely deployed transport protocol, TCP, was designed based on the assumption that packet losses are because of congestion (i.e. buffer overflow), and loss rates and delays are reasonably low. This assumption becomes invalid over wireless links when loss rates and delays are high, and packet losses may be due to wireless transmission errors or mobility handoffs, which are exogenous, i.e. independent of load or TCP's sending rate. By mistakenly backing off to such wireless errors, which may be temporary, TCP throughput unnecessarily decreases. To overcome this limitation, in-line devices ("middleboxes"), often called Performance Enhancing Proxies (PEP), have been developed to "hide" wireless losses or delays from the TCP sender. For example, hiding of wireless losses is usually accomplished by employing a different transport (error control) protocol over the wireless segment so as to recover wireless losses quickly before the TCP sender times out.



Exercises:

1) Assume we modify the TCP AIMD algorithm and instead use AIAD, i.e. Additive Increase Additive Decrease. So, when no congestion is observed, a source increases its window size by 1 packet every round-trip time (RTT). Whenever congestion is observed, a source decreases its window size by 1. Given two AIAD sources sharing the same bottleneck and experiencing the same RTT, do they converge to a fair and efficient rate allocation? Support your answer graphically by showing the trajectories of the two windows assuming a synchronized model where the windows are adapted at the same time instants.

2) For the following statements, either circle your choice, or fill in the blanks between parentheses:
   (a) The effective bandwidth for a source, needed to satisfy some performance

requirement, (increases OR decreases) as the traffic burstiness of the source decreases.

(b) Given the ideal transmission window size is C × D, where C is the bottleneck capacity and D is the round-trip propagation delay, the buffer size should be set to ( ) to make the AIMD operation of TCP efficient (i.e., 100% link utilization).

3) If a TCP-Tahoe source has a packet loss whenever its window size is 16, what is the range in which its window size oscillates in steady state? If the round-trip time (RTT) is 1, and the optimal window size is 8 (i.e. optimal in terms of 100% utilization), what is the utilization of the bottleneck link when the source has a window size *w*? Use this to compute the average utilization of the bottleneck if it only carries a single source. Recall TCP-Tahoe does not implement fast recovery.

4) Consider a FCFS queue of maximum size K packets to which packets arrive according to a Poisson process of rate λ. A packet is served for an exponentially distributed time with average $1/\mu$.

(a) Draw the steady-state transition diagram of the corresponding Markov chain, and solve for the steady-state probability of being in the different states.

(b) Write down expressions for the throughput and average packet delay. (You don't need to solve for the final closed-form equations.)

(c) How large does K have to be so that the probability of buffer overflow does not exceed X? (You don't need to solve for the final closed-form equation.)