Concurrent Computing (Computer Networks)

Daniel Page

Department of Computer Science,
University Of Bristol,
Merchant Venturers Building,
Woodland Road,
Bristol, BS8 1UB. UK.
⟨Daniel.Page@bristol.ac.uk⟩

March 8, 2016

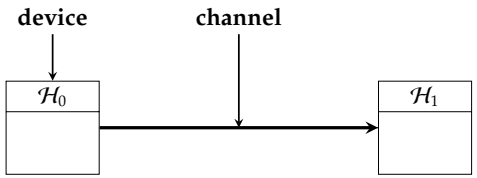
Keep in mind there are *two* PDFs available (of which this is the latter):

1. a PDF of examinable material used as lecture slides, and
2. a PDF of non-examinable, extra material:
   - the associated notes page may be pre-populated with extra, written explaination of material covered in lecture(s), plus
   - anything with a "grey'ed out" header/footer represents extra material which is useful and/or interesting but out of scope (and hence not covered).

Notes:

Notes:

- We can *already* form a simple **point-to-point** communication channel

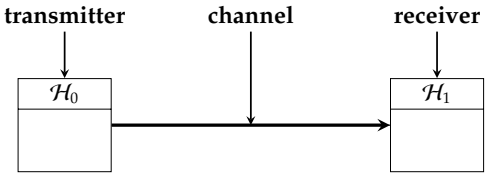


using TIA-232-F.

Notes:

COMS20001 lecture: week #19

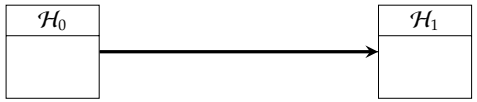
- We can *already* form a simple **point-to-point** communication channel



using TIA-232-F.

Notes:

- We can *already* form a simple **point-to-point** communication channel



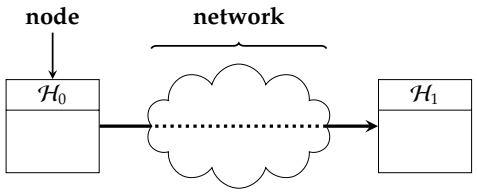
using TIA-232-F, st.

- Good:
  - modular wrt. communication medium and protocol,
  - uses standardised components,
  - ...
- Bad:
  - well defined, but quite limited functionality,
  - the organisation of components is fixed,
  - there are limits wrt. physical locality,
  - ...

Notes:

COMS20001 lecture: week #19

- We can *already* form a simple **point-to-point** communication channel



using TIA-232-F.

- Challenge: expand our remit to use of a **computer network** ...

Notes:

- ... *how*?

Notes:

- One of the challenges involved in specifying a precise set of requirements is that the network might be (simultaneously) used by a diverse population of users and wrt. a diverse range of applications; even then, *their* requirements differ from whoever is tasked with designing or maintaining the network. The resulting tension is illustrated by issues such as **network neutrality**, e.g.,
  - network users want a high quality of service,
  - (some) network operators want to extract economic benefit from investment in and operation of their infrastructure, applying non-uniform treatment of network traffic.

  This example aside, resolving this tension (if that is even possible) more generally demands a high degree of (technical) flexibility and modularity: if the network design can support diversity from a technical perspective, (out of scope) policy can then dictate use to suit.

COMS20001 lecture: week #19

- ... *how*?
  1. Requirements: what do we expect from a network?
     - supports a high degree of connectivity,
     - allows inter-operation between heterogeneous components,
     - uses appropriate level of abstraction to provide useful functionality,
     - satisfies any relevant quality metrics (e.g., efficiency, reliability),
     - can be (dynamically) scaled wrt. components and usage.

Notes:

- One of the challenges involved in specifying a precise set of requirements is that the network might be (simultaneously) used by a diverse population of users and wrt. a diverse range of applications; even then, *their* requirements differ from whoever is tasked with designing or maintaining the network. The resulting tension is illustrated by issues such as **network neutrality**, e.g.,
  - network users want a high quality of service,
  - (some) network operators want to extract economic benefit from investment in and operation of their infrastructure, applying non-uniform treatment of network traffic.

  This example aside, resolving this tension (if that is even possible) more generally demands a high degree of (technical) flexibility and modularity: if the network design can support diversity from a technical perspective, (out of scope) policy can then dictate use to suit.

Daniel Page ⟨daniel.page@bristol.ac.uk⟩
Concurrent Computing (Computer Networks) git # 2a1cd85 @ 2016-03-05 University of BRISTOL

- ... *how*?

2. Architecture: how should we approach the design of a network?
   - (physical or logical) **topology**, i.e., the structure of components in the network, plus
   - control i.e., how do we use those components to communicate, and
   - standardisation, i.e., how do we ensure components which communicate can **inter-operate**.

Notes:

- One of the challenges involved in specifying a precise set of requirements is that the network might be (simultaneously) used by a diverse population of users and wrt. a diverse range of applications; even then, *their* requirements differ from whoever is tasked with designing or maintaining the network. The resulting tension is illustrated by issues such as **network neutrality**, e.g.,
  - network users want a high quality of service,
  - (some) network operators want to extract economic benefit from investment in and operation of their infrastructure, applying non-uniform treatment of network traffic.

  This example aside, resolving this tension (if that is even possible) more generally demands a high degree of (technical) flexibility and modularity: if the network design can support diversity from a technical perspective, (out of scope) policy can then dictate use to suit.

Problem #1: topology (1)

- Idea #1: *fully*-connected, st. $\mathcal{H}_i$ is connected to $\mathcal{H}_j$ for $j \in S = \{0, 1, \ldots n-1\} \setminus \{i\}$.

- Note that access to a given channel may be

1. **dedicated**, or
2. **shared**, implying a need to control access but also the posibility to **broadcast** to *multiple* receiving nodes.

Notes:

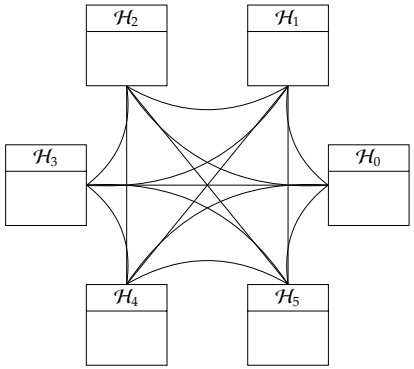
- Once shared channels are introduced, we can distinguish between two types of transmission: broadcast (where all hosts can receive the transmission, and which is enabled by either a shared wired or wireless channel), and point-to-point (where the other end-point can receive the transmission, but no other host can).
- A channel with shared access, e.g., in the bus topology, is may be termed **multi-point** (cf. point-to-point) or **multi-drop**.
- Although the lines drawn in such diagrams hint at wired channels, it is important to keep in mind they capture *any* communication medium. A bus topology, for example, may also be used to describe most wireless networks: the shared, wired bus is now a shared radio frequency (which of course you cannot "see" in the same way, but still acts as the communication medium).
- It should be clear that using a fully connected topology can quickly become unfeasible wrt. scale. The obvious example is for the full-connected mesh: for n nodes we need

$$\frac{n \cdot (n-1)}{2}$$

  duplex connections (or twice that for simplex connections), which obviously grows very quickly as n grows.

Problem #1: topology (1)

- Idea #1: *fully*-connected, st. $\mathcal{H}_i$ is connected to $\mathcal{H}_j$ for $j \in S = \{0, 1, \ldots n-1\} \setminus \{i\}$.
- Example: a **mesh topology**, i.e.,



- Note that access to a given channel may be
  1. **dedicated**, or
  2. **shared**, implying a need to control access but also the posibility to **broadcast** to *multiple* receiving nodes.

Notes:

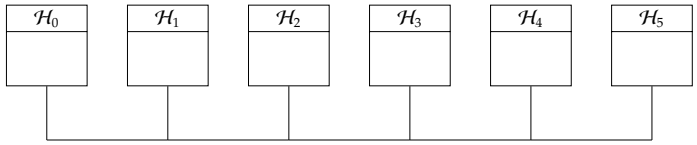
- Once shared channels are introduced, we can distinguish between two types of transmission: broadcast (where all hosts can receive the transmission, and which is enabled by either a shared wired or wireless channel), and point-to-point (where the other end-point can receive the transmission, but no other host can).
- A channel with shared access, e.g., in the bus topology, is may be termed **multi-point** (cf. point-to-point) or **multi-drop**.
- Although the lines drawn in such diagrams hint at wired channels, it is important to keep in mind they capture *any* communication medium. A bus topology, for example, may also be used to describe most wireless networks: the shared, wired bus is now a shared radio frequency (which of course you cannot "see" in the same way, but still acts as the communication medium).
- It should be clear that using a fully connected topology can quickly become unfeasible wrt. scale. The obvious example is for the full-connected mesh: for n nodes we need

$$\frac{n \cdot (n-1)}{2}$$

duplex connections (or twice that for simplex connections), which obviously grows very quickly as n grows.

Problem #1: topology (1)

- Idea #1: *fully*-connected, st. $\mathcal{H}_i$ is connected to $\mathcal{H}_j$ for $j \in S = \{0, 1, \ldots n-1\} \setminus \{i\}$.
- Example: a **bus topology**, i.e.,



- Note that access to a given channel may be
  1. **dedicated**, or
  2. **shared**, implying a need to control access but also the posibility to **broadcast** to *multiple* receiving nodes.

Daniel Page ⟨daniel.page@bristol.ac.uk⟩
Concurrent Computing (Computer Networks) git # 2a1cd85 @ 2016-03-05 University of BRISTOL

Notes:

- Once shared channels are introduced, we can distinguish between two types of transmission: broadcast (where all hosts can receive the transmission, and which is enabled by either a shared wired or wireless channel), and point-to-point (where the other end-point can receive the transmission, but no other host can).
- A channel with shared access, e.g., in the bus topology, is may be termed **multi-point** (cf. point-to-point) or **multi-drop**.
- Although the lines drawn in such diagrams hint at wired channels, it is important to keep in mind they capture *any* communication medium. A bus topology, for example, may also be used to describe most wireless networks: the shared, wired bus is now a shared radio frequency (which of course you cannot "see" in the same way, but still acts as the communication medium).
- It should be clear that using a fully connected topology can quickly become unfeasible wrt. scale. The obvious example is for the full-connected mesh: for n nodes we need

$$\frac{n \cdot (n-1)}{2}$$

duplex connections (or twice that for simplex connections), which obviously grows very quickly as n grows.

Problem #1: topology (2)

- Idea #2: *partially*-connected, st. $\mathcal{H}_i$ is connected to $\mathcal{H}_j$ for $j \in S \subset \{0, 1, \ldots n-1\} \setminus \{i\}$.

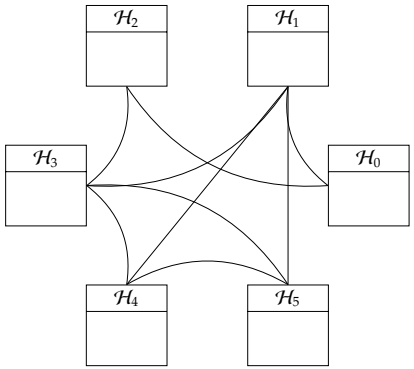
- Note that connectivity may now be either
  1. **direct**, or
  2. **indirect**, implying a need for intermediate **hops**, e.g., as realised via **store-and-forward** by (intermediate) switching nodes.

Notes:

- Another way to describe an indirect connection is to say that end-to-end *physical* connectivity may be discontinuous, even if a *logical* connection is maintained between the end-points.

Problem #1: topology (2)

- Idea #2: *partially*-connected, st. $\mathcal{H}_i$ is connected to $\mathcal{H}_j$ for $j \in S \subset \{0, 1, \ldots n-1\} \setminus \{i\}$.
- Example: a **mesh topology**, i.e.,



- Note that connectivity may now be either
  1. **direct**, or
  2. **indirect**, implying a need for intermediate **hops**, e.g., as realised via **store-and-forward** by (intermediate) switching nodes.

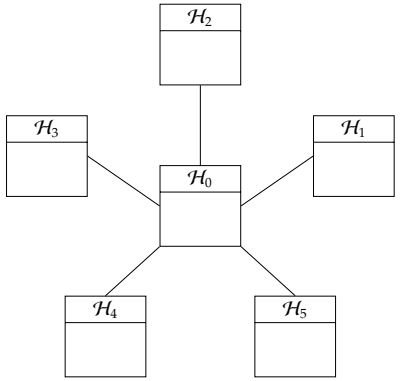
Daniel Page ⟨daniel.page@bristol.ac.uk⟩
Concurrent Computing (Computer Networks) git # 2a1cd85 @ 2016-03-05 University of BRISTOL

Notes:

- Another way to describe an indirect connection is to say that end-to-end *physical* connectivity may be discontinuous, even if a *logical* connection is maintained between the end-points.

Problem #1: topology (2)

- Idea #2: *partially*-connected, st. $\mathcal{H}_i$ is connected to $\mathcal{H}_j$ for $j \in S \subset \{0, 1, \ldots n-1\} \setminus \{i\}$.
- Example: a **star topology**, i.e.,



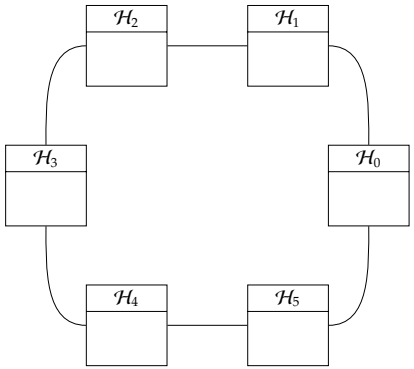
- Note that connectivity may now be either
  1. **direct**, or
  2. **indirect**, implying a need for intermediate **hops**, e.g., as realised via **store-and-forward** by (intermediate) switching nodes.

Notes:

- Another way to describe an indirect connection is to say that end-to-end *physical* connectivity may be discontinuous, even if a *logical* connection is maintained between the end-points.

Problem #1: topology (2)

- Idea #2: *partially*-connected, st. $\mathcal{H}_i$ is connected to $\mathcal{H}_j$ for $j \in S \subset \{0, 1, \ldots n-1\} \setminus \{i\}$.
- Example: a **ring topology**, i.e.,



- Note that connectivity may now be either
  1. **direct**, or
  2. **indirect**, implying a need for intermediate **hops**, e.g., as realised via **store-and-forward** by (intermediate) switching nodes.

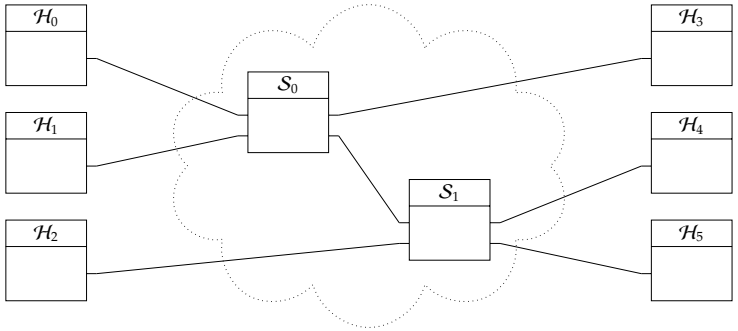
Daniel Page ⟨daniel.page@bristol.ac.uk⟩
Concurrent Computing (Computer Networks) git # 2a1cd85 @ 2016-03-05 University of BRISTOL

Notes:

- Another way to describe an indirect connection is to say that end-to-end *physical* connectivity may be discontinuous, even if a *logical* connection is maintained between the end-points.

Problem #1: topology (3)

- We can *generalise* partially connected network topologies



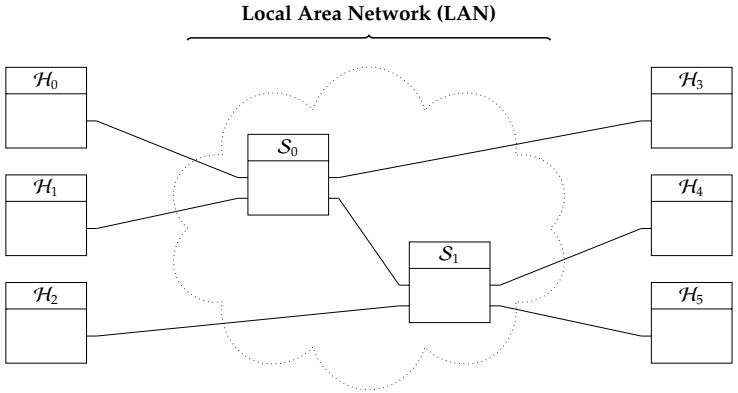
Notes:

- It is common say internet rather than inter-network, and distinguish *an* internet (note the capitalisation) from *the* Internet: the former relates to the concept of a network of networks, the latter to the specific, network of networks we term the Internet.
- The concept of a **virtual circuit** (or **virtual channel**) is a mixture of circuit and packet switching. The basic idea is to use packet switching st. it *appears* that a physical end-to-end connection exists, and hence there is a (virtual) end-to-end connection per circuit switching).
- The two different concepts form two different types of network **service model**. We see later than TCP and UDP offer users a
  - connection-less **datagram service model**, or
  - connection-based **virtual circuit model**

  respectively, on top of IPv4: the latter is based on packet switching, so essentially UDP and TCP offer an abstract model of the network (with some well defined functionality), which is implemented concretely using IPv4.
  For example, TCP adds a connection-based layer of abstraction to IP: in common with other examples, TCP realises this behaviour using a connection establishment phase. In other examples, this may be used to *reserve* resources in the network to ensure non-interference with other connections and hence a fixed service quality.

Problem #1: topology (3)

- We can *generalise* partially connected network topologies



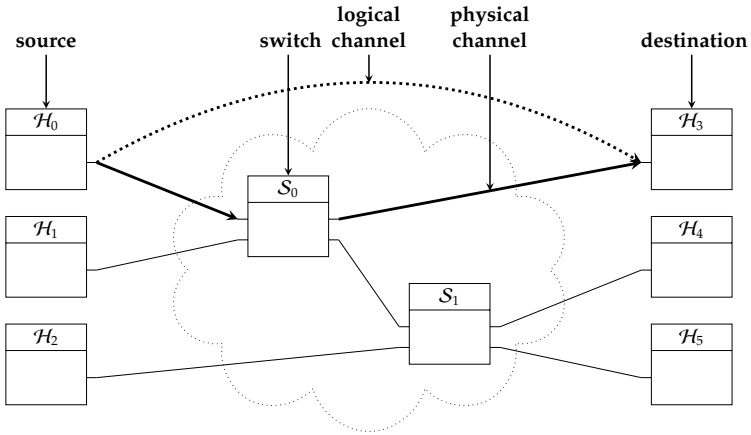
Notes:

- It is common say internet rather than inter-network, and distinguish *an* internet (note the capitalisation) from *the* Internet: the former relates to the concept of a network of networks, the latter to the specific, network of networks we term the Internet.
- The concept of a **virtual circuit** (or **virtual channel**) is a mixture of circuit and packet switching. The basic idea is to use packet switching st. it *appears* that a physical end-to-end connection exists, and hence there is a (virtual) end-to-end connection per circuit switching).
- The two different concepts form two different types of network **service model**. We see later than TCP and UDP offer users a
  - connection-less **datagram service model**, or
  - connection-based **virtual circuit model**

  respectively, on top of IPv4: the latter is based on packet switching, so essentially UDP and TCP offer an abstract model of the network (with some well defined functionality), which is implemented concretely using IPv4.
  For example, TCP adds a connection-based layer of abstraction to IP: in common with other examples, TCP realises this behaviour using a connection establishment phase. In other examples, this may be used to *reserve* resources in the network to ensure non-interference with other connections and hence a fixed service quality.

Problem #1: topology (3)

- We can *generalise* partially connected network topologies



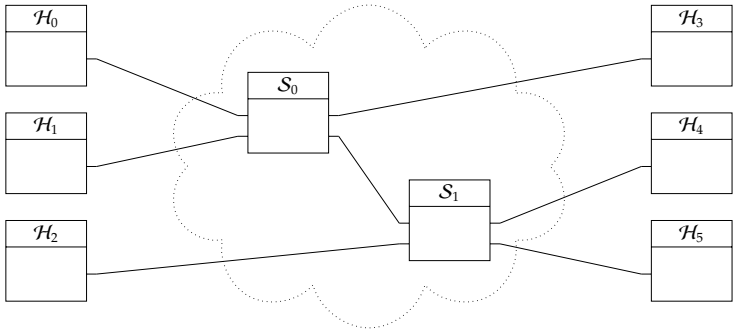
Notes:

- It is common say internet rather than inter-network, and distinguish *an* internet (note the capitalisation) from *the* Internet: the former relates to the concept of a network of networks, the latter to the specific, network of networks we term the Internet.
- The concept of a **virtual circuit** (or **virtual channel**) is a mixture of circuit and packet switching. The basic idea is to use packet switching st. it *appears* that a physical end-to-end connection exists, and hence there is a (virtual) end-to-end connection per circuit switching).
- The two different concepts form two different types of network **service model**. We see later than TCP and UDP offer users a
  - connection-less **datagram service model**, or
  - connection-based **virtual circuit model**

  respectively, on top of IPv4: the latter is based on packet switching, so essentially UDP and TCP offer an abstract model of the network (with some well defined functionality), which is implemented concretely using IPv4.
  For example, TCP adds a connection-based layer of abstraction to IP: in common with other examples, TCP realises this behaviour using a connection establishment phase. In other examples, this may be used to *reserve* resources in the network to ensure non-interference with other connections and hence a fixed service quality.

Problem #1: topology (3)

- We can *generalise* partially connected network topologies



leading to the concepts of

- **circuit switching** and
- **packet switching** (to support connection-based or connection-less channels).

Daniel Page ⟨daniel.page@bristol.ac.uk⟩
Concurrent Computing (Computer Networks) git # 2a1cd85 @ 2016-03-05 University of BRISTOL

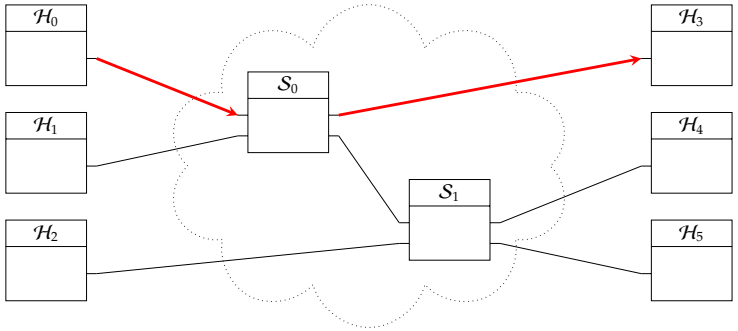
Notes:

- It is common say internet rather than inter-network, and distinguish *an* internet (note the capitalisation) from *the* Internet: the former relates to the concept of a network of networks, the latter to the specific, network of networks we term the Internet.
- The concept of a **virtual circuit** (or **virtual channel**) is a mixture of circuit and packet switching. The basic idea is to use packet switching st. it *appears* that a physical end-to-end connection exists, and hence there is a (virtual) end-to-end connection per circuit switching).
- The two different concepts form two different types of network **service model**. We see later than TCP and UDP offer users a
  - connection-less **datagram service model**, or
  - connection-based **virtual circuit model**

  respectively, on top of IPv4: the latter is based on packet switching, so essentially UDP and TCP offer an abstract model of the network (with some well defined functionality), which is implemented concretely using IPv4.
  For example, TCP adds a connection-based layer of abstraction to IP: in common with other examples, TCP realises this behaviour using a connection establishment phase. In other examples, this may be used to *reserve* resources in the network to ensure non-interference with other connections and hence a fixed service quality.

Problem #1: topology (3)

- We can *generalise* partially connected network topologies



leading to the concepts of

- **circuit switching** and
- **packet switching** (to support connection-based or connection-less channels).

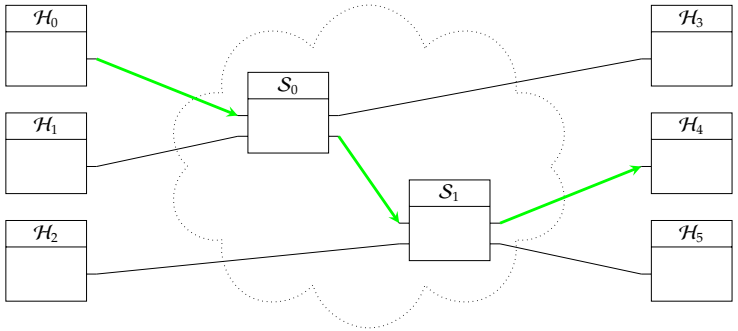
Notes:

- It is common say internet rather than inter-network, and distinguish *an* internet (note the capitalisation) from *the* Internet: the former relates to the concept of a network of networks, the latter to the specific, network of networks we term the Internet.
- The concept of a **virtual circuit** (or **virtual channel**) is a mixture of circuit and packet switching. The basic idea is to use packet switching st. it *appears* that a physical end-to-end connection exists, and hence there is a (virtual) end-to-end connection per circuit switching).
- The two different concepts form two different types of network **service model**. We see later than TCP and UDP offer users a
  - connection-less **datagram service model**, or
  - connection-based **virtual circuit model**

  respectively, on top of IPv4: the latter is based on packet switching, so essentially UDP and TCP offer an abstract model of the network (with some well defined functionality), which is implemented concretely using IPv4.
  For example, TCP adds a connection-based layer of abstraction to IP: in common with other examples, TCP realises this behaviour using a connection establishment phase. In other examples, this may be used to *reserve* resources in the network to ensure non-interference with other connections and hence a fixed service quality.

Problem #1: topology (3)

- We can *generalise* partially connected network topologies



leading to the concepts of

- **circuit switching** and
- **packet switching** (to support connection-based or connection-less channels).

Daniel Page ⟨daniel.page@bristol.ac.uk⟩
Concurrent Computing (Computer Networks) git # 2a1cd85 @ 2016-03-05 University of BRISTOL

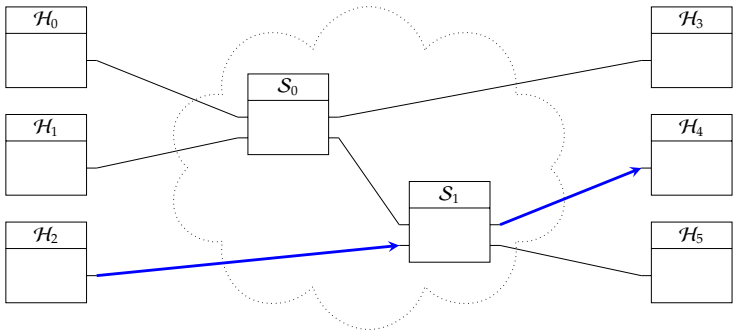
Notes:

- It is common say internet rather than inter-network, and distinguish *an* internet (note the capitalisation) from *the* Internet: the former relates to the concept of a network of networks, the latter to the specific, network of networks we term the Internet.
- The concept of a **virtual circuit** (or **virtual channel**) is a mixture of circuit and packet switching. The basic idea is to use packet switching st. it *appears* that a physical end-to-end connection exists, and hence there is a (virtual) end-to-end connection per circuit switching).
- The two different concepts form two different types of network **service model**. We see later than TCP and UDP offer users a
  - connection-less **datagram service model**, or
  - connection-based **virtual circuit model**

  respectively, on top of IPv4: the latter is based on packet switching, so essentially UDP and TCP offer an abstract model of the network (with some well defined functionality), which is implemented concretely using IPv4.
  For example, TCP adds a connection-based layer of abstraction to IP: in common with other examples, TCP realises this behaviour using a connection establishment phase. In other examples, this may be used to *reserve* resources in the network to ensure non-interference with other connections and hence a fixed service quality.

Problem #1: topology (3)

- We can *generalise* partially connected network topologies



leading to the concepts of

- **circuit switching** and
- **packet switching** (to support connection-based or connection-less channels).

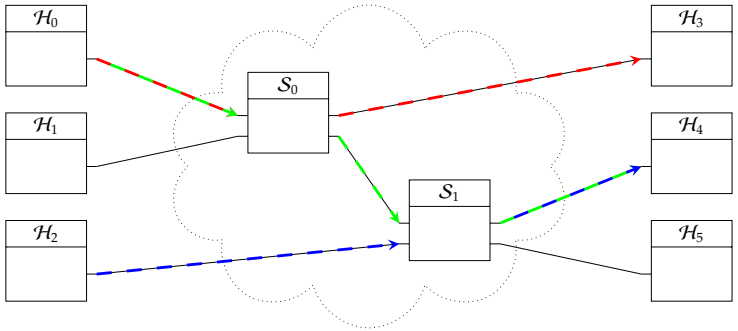
Notes:

- It is common say internet rather than inter-network, and distinguish *an* internet (note the capitalisation) from *the* Internet: the former relates to the concept of a network of networks, the latter to the specific, network of networks we term the Internet.
- The concept of a **virtual circuit** (or **virtual channel**) is a mixture of circuit and packet switching. The basic idea is to use packet switching st. it *appears* that a physical end-to-end connection exists, and hence there is a (virtual) end-to-end connection per circuit switching).
- The two different concepts form two different types of network **service model**. We see later than TCP and UDP offer users a
  - connection-less **datagram service model**, or
  - connection-based **virtual circuit model**

  respectively, on top of IPv4: the latter is based on packet switching, so essentially UDP and TCP offer an abstract model of the network (with some well defined functionality), which is implemented concretely using IPv4.
  For example, TCP adds a connection-based layer of abstraction to IP: in common with other examples, TCP realises this behaviour using a connection establishment phase. In other examples, this may be used to *reserve* resources in the network to ensure non-interference with other connections and hence a fixed service quality.

Problem #1: topology (3)

- We can *generalise* partially connected network topologies



leading to the concepts of

- **circuit switching** and
- **packet switching** (to support connection-based or connection-less channels).

Daniel Page ⟨daniel.page@bristol.ac.uk⟩
Concurrent Computing (Computer Networks) git # 2a1cd85 @ 2016-03-05 University of BRISTOL

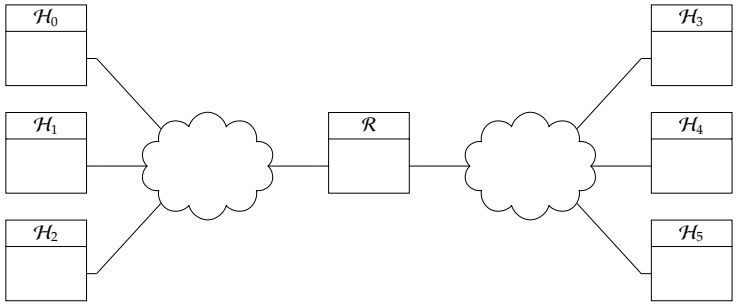
Notes:

- It is common say internet rather than inter-network, and distinguish *an* internet (note the capitalisation) from *the* Internet: the former relates to the concept of a network of networks, the latter to the specific, network of networks we term the Internet.
- The concept of a **virtual circuit** (or **virtual channel**) is a mixture of circuit and packet switching. The basic idea is to use packet switching st. it *appears* that a physical end-to-end connection exists, and hence there is a (virtual) end-to-end connection per circuit switching).
- The two different concepts form two different types of network **service model**. We see later than TCP and UDP offer users a
  - connection-less **datagram service model**, or
  - connection-based **virtual circuit model**

  respectively, on top of IPv4: the latter is based on packet switching, so essentially UDP and TCP offer an abstract model of the network (with some well defined functionality), which is implemented concretely using IPv4.
  For example, TCP adds a connection-based layer of abstraction to IP: in common with other examples, TCP realises this behaviour using a connection establishment phase. In other examples, this may be used to *reserve* resources in the network to ensure non-interference with other connections and hence a fixed service quality.

Problem #1: topology (3)

- We can *generalise* partially connected network topologies



then, finally, noting

- a network provides connectivity between nodes, while
- an *inter*-network connects networks themselves together.

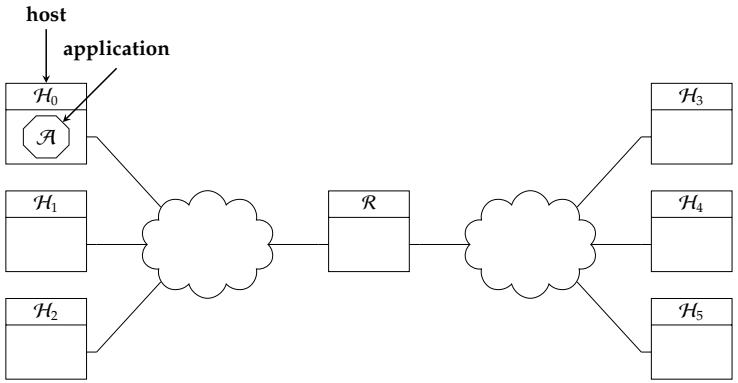
Notes:

- It is common say internet rather than inter-network, and distinguish *an* internet (note the capitalisation) from *the* Internet: the former relates to the concept of a network of networks, the latter to the specific, network of networks we term the Internet.
- The concept of a **virtual circuit** (or **virtual channel**) is a mixture of circuit and packet switching. The basic idea is to use packet switching st. it *appears* that a physical end-to-end connection exists, and hence there is a (virtual) end-to-end connection per circuit switching).
- The two different concepts form two different types of network **service model**. We see later than TCP and UDP offer users a
  - connection-less **datagram service model**, or
  - connection-based **virtual circuit model**

  respectively, on top of IPv4: the latter is based on packet switching, so essentially UDP and TCP offer an abstract model of the network (with some well defined functionality), which is implemented concretely using IPv4.
  For example, TCP adds a connection-based layer of abstraction to IP: in common with other examples, TCP realises this behaviour using a connection establishment phase. In other examples, this may be used to *reserve* resources in the network to ensure non-interference with other connections and hence a fixed service quality.

Problem #1: topology (3)

- We can *generalise* partially connected network topologies



then, finally, noting

- a network provides connectivity between nodes, while
- an *inter*-network connects networks themselves together.

Daniel Page ⟨daniel.page@bristol.ac.uk⟩
Concurrent Computing (Computer Networks) git # 2a1cd85 @ 2016-03-05 University of BRISTOL

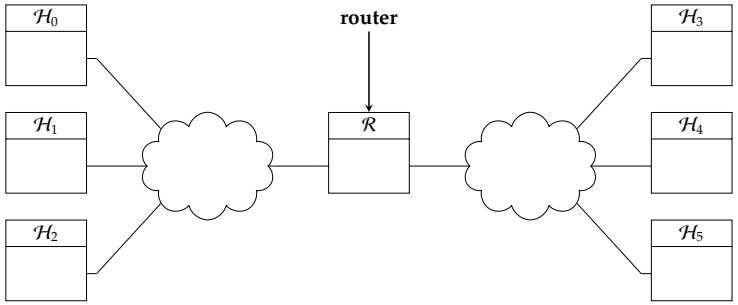
Notes:

- It is common say internet rather than inter-network, and distinguish *an* internet (note the capitalisation) from *the* Internet: the former relates to the concept of a network of networks, the latter to the specific, network of networks we term the Internet.
- The concept of a **virtual circuit** (or **virtual channel**) is a mixture of circuit and packet switching. The basic idea is to use packet switching st. it *appears* that a physical end-to-end connection exists, and hence there is a (virtual) end-to-end connection per circuit switching).
- The two different concepts form two different types of network **service model**. We see later than TCP and UDP offer users a
  - connection-less **datagram service model**, or
  - connection-based **virtual circuit model**

  respectively, on top of IPv4: the latter is based on packet switching, so essentially UDP and TCP offer an abstract model of the network (with some well defined functionality), which is implemented concretely using IPv4.
  For example, TCP adds a connection-based layer of abstraction to IP: in common with other examples, TCP realises this behaviour using a connection establishment phase. In other examples, this may be used to *reserve* resources in the network to ensure non-interference with other connections and hence a fixed service quality.

Problem #1: topology (3)

- We can *generalise* partially connected network topologies



then, finally, noting

- a network provides connectivity between nodes, while
- an *inter*-network connects networks themselves together.

Notes:

- It is common say internet rather than inter-network, and distinguish *an* internet (note the capitalisation) from *the* Internet: the former relates to the concept of a network of networks, the latter to the specific, network of networks we term the Internet.
- The concept of a **virtual circuit** (or **virtual channel**) is a mixture of circuit and packet switching. The basic idea is to use packet switching st. it *appears* that a physical end-to-end connection exists, and hence there is a (virtual) end-to-end connection per circuit switching).
- The two different concepts form two different types of network **service model**. We see later than TCP and UDP offer users a
  - connection-less **datagram service model**, or
  - connection-based **virtual circuit model**

  respectively, on top of IPv4: the latter is based on packet switching, so essentially UDP and TCP offer an abstract model of the network (with some well defined functionality), which is implemented concretely using IPv4.
  For example, TCP adds a connection-based layer of abstraction to IP: in common with other examples, TCP realises this behaviour using a connection establishment phase. In other examples, this may be used to *reserve* resources in the network to ensure non-interference with other connections and hence a fixed service quality.

Problem #1: topology (4)

- Analogy #1: the (circa 1940) telephone network

(roughly) explains circuit switching:

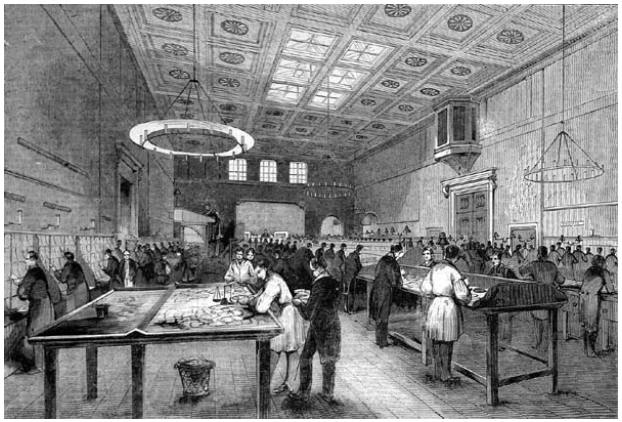
- each telephone has a number,
- an operator manages connection between source and destination,
- each call has dedicated access to required components,
- connection is (resp. telephone numbers are) hierarchical.

http://en.wikipedia.org/wiki/File:Photograph_of_Women_Working_at_a_Bell_System_Telephone_Switchboard_(3660047829).jpg

Notes:

Problem #1: topology (5)

- Analogy #2: the postal network



(roughly) explains packet switching:

- each house has an address,
- a sorting office manages delivery between source and destination,
- each letter shares access to required components (e.g., post man),
- delivery is (resp. addresses are) hierarchical.

http://en.wikipedia.org/wiki/File:GPO_Inland_letter_office_ILN_1845.jpg

Notes:

- Actually, this analogy isn't *that* good because the normal use-case is for many users to each send one letter: to understand packet switching from the perspective of that user, it's better to think of them sending a giant letter which is cut into smaller parts (due to the new UK model of changing per size with a weight limit, rather than just weight alone) each sent separately.

Problem #1: topology (6)

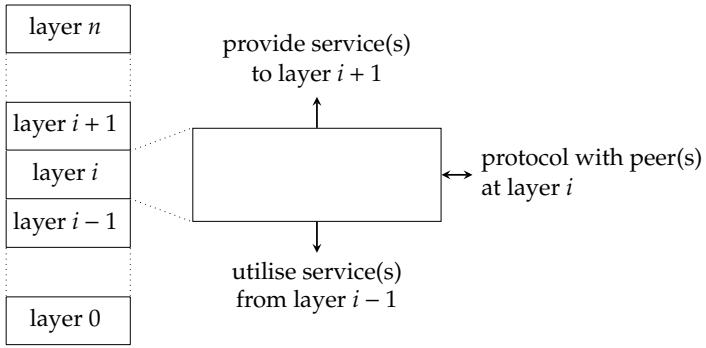
- An inter-network architecture seems a good approach!
  - Good:
    - scalable by virtue of a flexible and modular design, *and*
    - offers resilience against failure.
  - Bad: we need to cope with the fact
    - ideally we support a multiplicity of communication mediums and protocols, and share access to both,
    - each host needs a globally unique **address**, and
    - router(s) need to know or discover how to communicate (or **route**) data from one host to another

  the difficulty of which is enhanced by a need to avoid impact on scalability (e.g., avoid centralised solutions).

Notes:

Problem #2: control (1)

- Step #1: rather than use one monolithic protocol, decompose control, e.g.,



and hence provide various advantages stemming from modularity ...

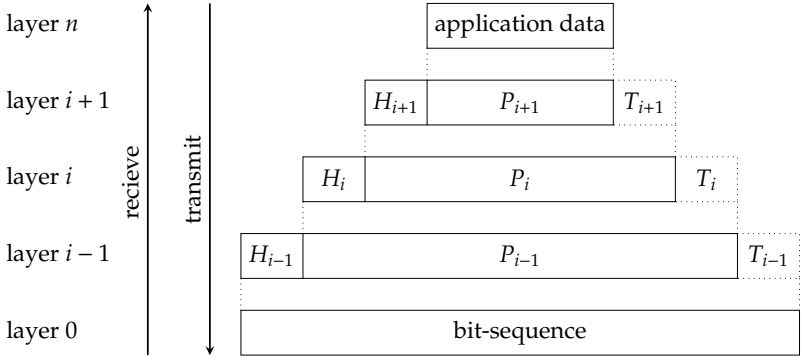
Notes:

- Some might be obvious, but the advantages include (at least) the fact it
  - provides a clean delineation of function, and is hence easier to understand (even at scale),
  - maximises compatibility and hence inter-operation (e.g., between different vendors), and
  - minimises disruption when a layer is changed somehow (e.g., to specialise or enhance functionality)

  There are, arguably, *disadvantages* as well. Common criticisms of too rigid an application of layering include the fact it can add overhead vs. a more streamlined, special-purpose alternative, plus
  - one layer may be tempted to duplicate functionality which, for whatever reason, is not guaranteed elsewhere (e.g., error detection), and
  - the fact that often it is *useful* for one layer to have access to information or state within another.
- It might be obvious, or maybe not, but there are two interfaces defined by each layer: the **service interface** used by protocols in other layers, and the **peer interface** used by protocols in the same layer (including the same protocol on another host).
- The term peer is intentional, stressing the fact that two peer protocols are equivalent in terms of seniority: no centralised control is used to dictate their operation.

Problem #2: control (2)

- ... then support inter-layer (peer or otherwise) communication using **encapsulation**:



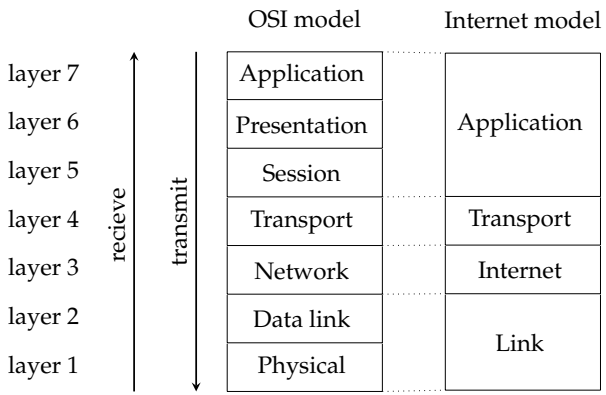
Notes:

- Each H_i is a **header** and T_i a **trailer** (or **footer**) that **frame** a **payload** P_i (i.e., the data). Note that:
  - data passed from a service at layer i to that at layer $i-1$ is opaque,
  - each header *must* contain enough control information to then be correctly interpreted by a peer protocol (also at layer i), *but*
  - the addition of headers (and trailers) implies overhead!
- Two examples of control information that might be included in a header are **addresses** and **multiplexing keys**. Using a more concrete context, this claim is easier to justify: at the IP layer,
  - an IP packet must include an address so a host can know whether to accept it or decide how to route it,
  - an IP packet must include a (de)multiplexing key so a host which accepts it knows which higher-level protocol (e.g., TCP or UDP) should perform the next stage of processing.
- Formally, there are *two* forms of encapsulation:
  1. a **Service Data Unit (SDU)** is how *unstructured* data is communicated across the service interface, while
  2. a **Protocol Data Unit (PDU)** is how *structured* data is communicated across the peer peer interface.

  This may seem pedantic (and it *is* common to use PDU for both cases). However, it stresses the fact data provided by one service as input for another is (by design) opaque: the second service cannot "look inside" and interpret the data, since this breaks the abstraction offered by the layered model. In contrast, for two peer protocols it *must* be the case that both can interpret the data communicated, since this is what enables a communication channel between them.

Problem #2: control (3)

- Step #2: specify (abstract) layers we need, e.g.,



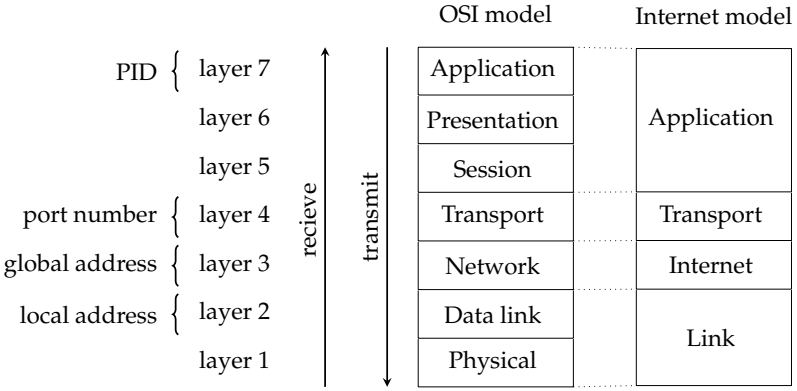
Notes:

- Either model attempts to capture and organise protocols, services and the interfaces to them. It might be obvious, or maybe not, but notice that :
  1. the vertical layers represent combination of protocols through their service interfaces, but
  2. the protocols themselves are arranged horizontally: even instances on separate hosts are peers.
- The term **Internet model** is not universal: you may see alternatives such as **DoD model** (hinting at the fact the design has ARPA-funded roots), or **TCP/IP model/suite** (hinting at the fact TCP and IP are core protocols in real instantiations).
- A brief overview of layers in the OSI model, with roughly the same applying (after concatenating the descriptions) to the Internet model, is as follows:
  - the physical layer transmits bits over the communication medium,
  - the data link layer transmits structured units of data, namely **frames**, via the physical layer (assuming a *direct* channel),
  - the network layer transmits structured units of data, namely **packets**, via the data link layer (supporting *indirect* channels, e.g., an inter-network topology),
  - the transport layer adds various end-to-end transmission services, such as reliability, to raw transmission,
  - the session layer acts to manage the prolonged (vs. a "one-shot" transmit or receive) interaction with lower layers,
  - the presentation layer abstracts issues of data representation (e.g., the compression and/or encryption of data), and
  - the application layer interacts with the user to provide whatever functionality is required.

  Note, however, that some (e.g., the session layer) are slightly vague and may seem under-defined without a more concrete example; the problem is, examples are hard to give because we normally prefer the Internet model!
- Where a number is used to refer to a layer, this usually relates to the OSI model: if we say "at layer-2" for example, we usually mean "at the data link layer".
- It is important to view where each layer is implemented (left) as a typical approach not the *only* approach. The lower-layers are *typically* implemented by the OS to ensure high-performance and protection (of shared hardware resources, via a device driver), but this may differ on an embedded system: there might not even *be* an OS, so the same layers may equally be executed in user space.

Problem #2: control (3)

- Step #2: specify (abstract) layers we need, e.g.,



Notes:

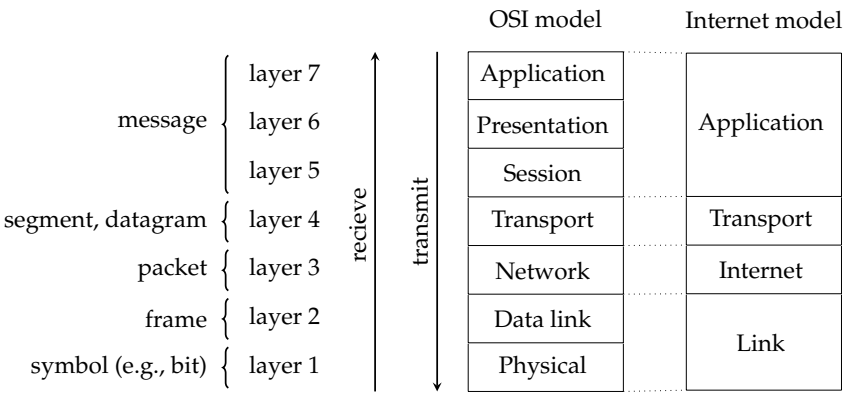
- Either model attempts to capture and organise protocols, services and the interfaces to them. It might be obvious, or maybe not, but notice that :
  1. the vertical layers represent combination of protocols through their service interfaces, but
  2. the protocols themselves are arranged horizontally: even instances on separate hosts are peers.
- The term **Internet model** is not universal: you may see alternatives such as **DoD model** (hinting at the fact the design has ARPA-funded roots), or **TCP/IP model/suite** (hinting at the fact TCP and IP are core protocols in real instantiations).
- A brief overview of layers in the OSI model, with roughly the same applying (after concatenating the descriptions) to the Internet model, is as follows:
  - the physical layer transmits bits over the communication medium,
  - the data link layer transmits structured units of data, namely **frames**, via the physical layer (assuming a *direct* channel),
  - the network layer transmits structured units of data, namely **packets**, via the data link layer (supporting *indirect* channels, e.g., an inter-network topology),
  - the transport layer adds various end-to-end transmission services, such as reliability, to raw transmission,
  - the session layer acts to manage the prolonged (vs. a "one-shot" transmit or receive) interaction with lower layers,
  - the presentation layer abstracts issues of data representation (e.g., the compression and/or encryption of data), and
  - the application layer interacts with the user to provide whatever functionality is required.

  Note, however, that some (e.g., the session layer) are slightly vague and may seem under-defined without a more concrete example; the problem is, examples are hard to give because we normally prefer the Internet model!
- Where a number is used to refer to a layer, this usually relates to the OSI model: if we say "at layer-2" for example, we usually mean "at the data link layer".
- It is important to view where each layer is implemented (left) as a typical approach not the *only* approach. The lower-layers are *typically* implemented by the OS to ensure high-performance and protection (of shared hardware resources, via a device driver), but this may differ on an embedded system: there might not even *be* an OS, so the same layers may equally be executed in user space.

Daniel Page ⟨daniel.page@bristol.ac.uk⟩
Concurrent Computing (Computer Networks) git # 2a1cd85 @ 2016-03-05 University of BRISTOL

Problem #2: control (3)

- Step #2: specify (abstract) layers we need, e.g.,



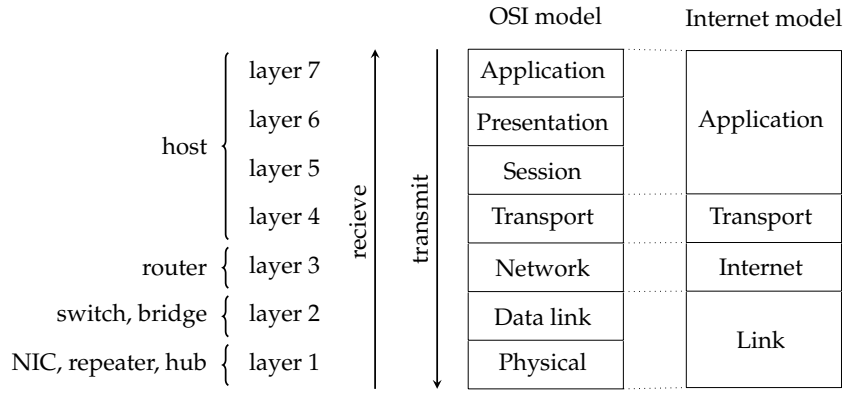
Notes:

- Either model attempts to capture and organise protocols, services and the interfaces to them. It might be obvious, or maybe not, but notice that :
  1. the vertical layers represent combination of protocols through their service interfaces, but
  2. the protocols themselves are arranged horizontally: even instances on separate hosts are peers.
- The term **Internet model** is not universal: you may see alternatives such as **DoD model** (hinting at the fact the design has ARPA-funded roots), or **TCP/IP model/suite** (hinting at the fact TCP and IP are core protocols in real instantiations).
- A brief overview of layers in the OSI model, with roughly the same applying (after concatenating the descriptions) to the Internet model, is as follows:
  - the physical layer transmits bits over the communication medium,
  - the data link layer transmits structured units of data, namely **frames**, via the physical layer (assuming a *direct* channel),
  - the network layer transmits structured units of data, namely **packets**, via the data link layer (supporting *indirect* channels, e.g., an inter-network topology),
  - the transport layer adds various end-to-end transmission services, such as reliability, to raw transmission,
  - the session layer acts to manage the prolonged (vs. a "one-shot" transmit or receive) interaction with lower layers,
  - the presentation layer abstracts issues of data representation (e.g., the compression and/or encryption of data), and
  - the application layer interacts with the user to provide whatever functionality is required.

  Note, however, that some (e.g., the session layer) are slightly vague and may seem under-defined without a more concrete example; the problem is, examples are hard to give because we normally prefer the Internet model!
- Where a number is used to refer to a layer, this usually relates to the OSI model: if we say "at layer-2" for example, we usually mean "at the data link layer".
- It is important to view where each layer is implemented (left) as a typical approach not the *only* approach. The lower-layers are *typically* implemented by the OS to ensure high-performance and protection (of shared hardware resources, via a device driver), but this may differ on an embedded system: there might not even *be* an OS, so the same layers may equally be executed in user space.

Problem #2: control (3)

- Step #2: specify (abstract) layers we need, e.g.,



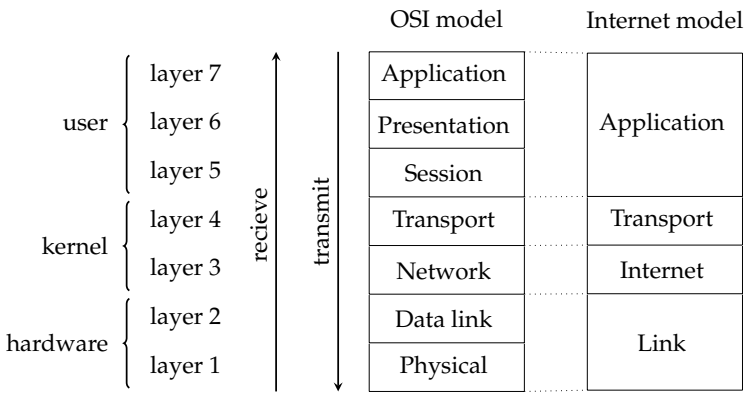
Notes:

- Either model attempts to capture and organise protocols, services and the interfaces to them. It might be obvious, or maybe not, but notice that :
  1. the vertical layers represent combination of protocols through their service interfaces, but
  2. the protocols themselves are arranged horizontally: even instances on separate hosts are peers.
- The term **Internet model** is not universal: you may see alternatives such as **DoD model** (hinting at the fact the design has ARPA-funded roots), or **TCP/IP model/suite** (hinting at the fact TCP and IP are core protocols in real instantiations).
- A brief overview of layers in the OSI model, with roughly the same applying (after concatenating the descriptions) to the Internet model, is as follows:
  - the physical layer transmits bits over the communication medium,
  - the data link layer transmits structured units of data, namely **frames**, via the physical layer (assuming a *direct* channel),
  - the network layer transmits structured units of data, namely **packets**, via the data link layer (supporting *indirect* channels, e.g., an inter-network topology),
  - the transport layer adds various end-to-end transmission services, such as reliability, to raw transmission,
  - the session layer acts to manage the prolonged (vs. a "one-shot" transmit or receive) interaction with lower layers,
  - the presentation layer abstracts issues of data representation (e.g., the compression and/or encryption of data), and
  - the application layer interacts with the user to provide whatever functionality is required.

  Note, however, that some (e.g., the session layer) are slightly vague and may seem under-defined without a more concrete example; the problem is, examples are hard to give because we normally prefer the Internet model!
- Where a number is used to refer to a layer, this usually relates to the OSI model: if we say "at layer-2" for example, we usually mean "at the data link layer".
- It is important to view where each layer is implemented (left) as a typical approach not the *only* approach. The lower-layers are *typically* implemented by the OS to ensure high-performance and protection (of shared hardware resources, via a device driver), but this may differ on an embedded system: there might not even *be* an OS, so the same layers may equally be executed in user space.

Problem #2: control (3)

- Step #2: specify (abstract) layers we need, e.g.,



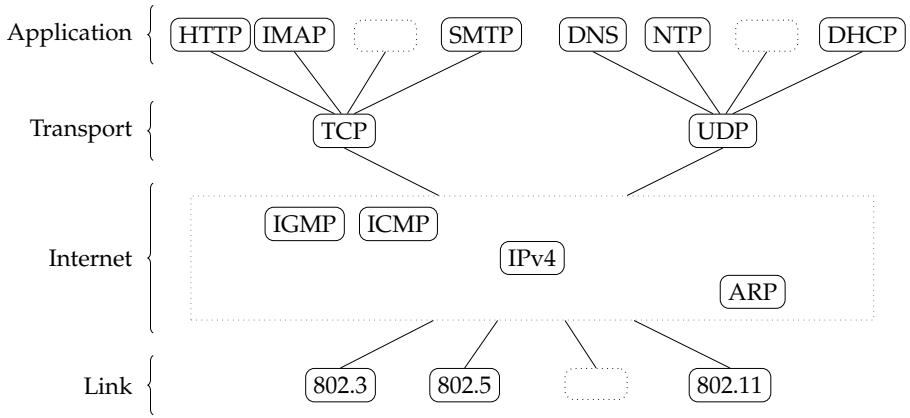
Notes:

- Either model attempts to capture and organise protocols, services and the interfaces to them. It might be obvious, or maybe not, but notice that :
  1. the vertical layers represent combination of protocols through their service interfaces, but
  2. the protocols themselves are arranged horizontally: even instances on separate hosts are peers.
- The term **Internet model** is not universal: you may see alternatives such as **DoD model** (hinting at the fact the design has ARPA-funded roots), or **TCP/IP model/suite** (hinting at the fact TCP and IP are core protocols in real instantiations).
- A brief overview of layers in the OSI model, with roughly the same applying (after concatenating the descriptions) to the Internet model, is as follows:
  - the physical layer transmits bits over the communication medium,
  - the data link layer transmits structured units of data, namely **frames**, via the physical layer (assuming a *direct* channel),
  - the network layer transmits structured units of data, namely **packets**, via the data link layer (supporting *indirect* channels, e.g., an inter-network topology),
  - the transport layer adds various end-to-end transmission services, such as reliability, to raw transmission,
  - the session layer acts to manage the prolonged (vs. a "one-shot" transmit or receive) interaction with lower layers,
  - the presentation layer abstracts issues of data representation (e.g., the compression and/or encryption of data), and
  - the application layer interacts with the user to provide whatever functionality is required.

  Note, however, that some (e.g., the session layer) are slightly vague and may seem under-defined without a more concrete example; the problem is, examples are hard to give because we normally prefer the Internet model!
- Where a number is used to refer to a layer, this usually relates to the OSI model: if we say "at layer-2" for example, we usually mean "at the data link layer".
- It is important to view where each layer is implemented (left) as a typical approach not the *only* approach. The lower-layers are *typically* implemented by the OS to ensure high-performance and protection (of shared hardware resources, via a device driver), but this may differ on an embedded system: there might not even *be* an OS, so the same layers may equally be executed in user space.

Problem #2: control (4)

- Step #3: populate layers with (concrete) protocols, e.g.,



noting that

- a **protocol graph** is an abstract description of how protocols fit into the layered model, whereas
- a **protocol stack** is a concrete implementation of *one* top-to-bottom combination of protocols.

Notes:

- When drawn like this, it should be clear that IP represents the bottleneck (for pessimists) or linchpin (for optimists) protocol: this is stressed by occasional use of the term **hourglass model**.
- This design has analogies elsewhere. Some compilers, for example, adopt a design that couples a common core to (many) optional front- and back-ends (for different languages and target platforms respectively).
- Although we explicitly do *not* cover IPv6, you can imaging extending a protocol graph like this to include it. The result becomes fairly complex, but roughly then includes *two* middle layers (IPv4 and IPv6) and some conditional dependencies: TCP can use either IPv4 or IPv6 transparently up to a point, for example, but some applications may need to be IP-aware wrt. addressing etc. and clearly there need to be IPv4 *and* IPv6 versions of protocols such as ICMP.
- Another way to describe a protocol graph versus a protocol stack is that the former dictates what a host *could* do (e.g., it could use UDP or TCP on top of IPv4), whereas the latter dictates what a host *does* do (i.e., to deal with some traffic on the network, it might employ a stack described by

  $$\text{HTTP} \rightarrow \text{TCP} \rightarrow \text{IPv4} \rightarrow \text{802.3}$$

  and hence is using TCP on top of IPv4 in this case).

- An effective computer network depends *fundamentally* on standardisation ...
- ... to inter-operate, *everything* needs to use
  1. standard protocols and interfaces, e.g., by the **Internet Engineering Task Force (IETF)**

     `http://www.ietf.org`
  2. standard identifiers etc. within them, e.g., by the **Internet Assigned Numbers Authority (IANA)**

     `http://www.iana.org`
  3. implementations that follow **Postel's Law**

Quote

TCP implementations will follow a general principle of robustness: be conservative in what you do, be liberal in what you accept from others.

– Postel [8, Section 2.10]

otherwise the "value" involved is diminished.

Notes:

- Metcalfe's Law suggests the "value" of an n node network is proportional to n^2, i.e., large networks (with more potential connectivity) are more valuable than smaller ones (with less).

- A central example is representation of data:

Quote

The convention in the documentation of Internet Protocols is to express numbers in decimal and to picture data in "big-endian" order [7].

The order of transmission of the header and data described in this document is resolved to the octet level. Whenever a diagram shows a group of octets, the order of transmission of those octets is the normal order in which they are read in English.

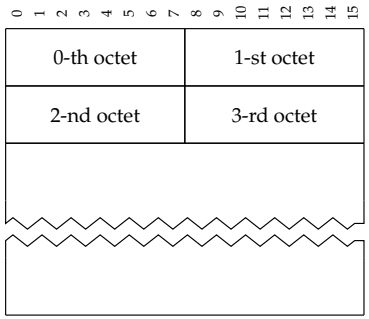
– Reynolds and Postel [9]

- Translation:
  1. an 8-bit **byte** is termed an **octet**,
  2. **network order** implies a **big-endian** ordering of octets *and* bits within them,
  3. transmission of data uses network order, meaning translation at either end-point may be required (to match processor endianness).

Notes:

- The term octet might seem (and basically is) slightly odd. Apparently the goal is to avoid confusion if a byte is *not* 8 bits (thus assuming that a byte is of abstract size, much like a word). Relevant examples of this case apparently *do* exist; an octet is *always* 8 bits, resolving any ambiguity (for those people with 7-bit bytes, but thereby also confusing everyone else).
- You may be familiar with `atoi` for ASCII to integer conversion; a range of similar(ish) functions, e.g., `htonl`, `htons`, (resp. `ntohl` and `ntohs`), can perform host-to-network (resp. network-to-host) conversion wrt. byte ordering.

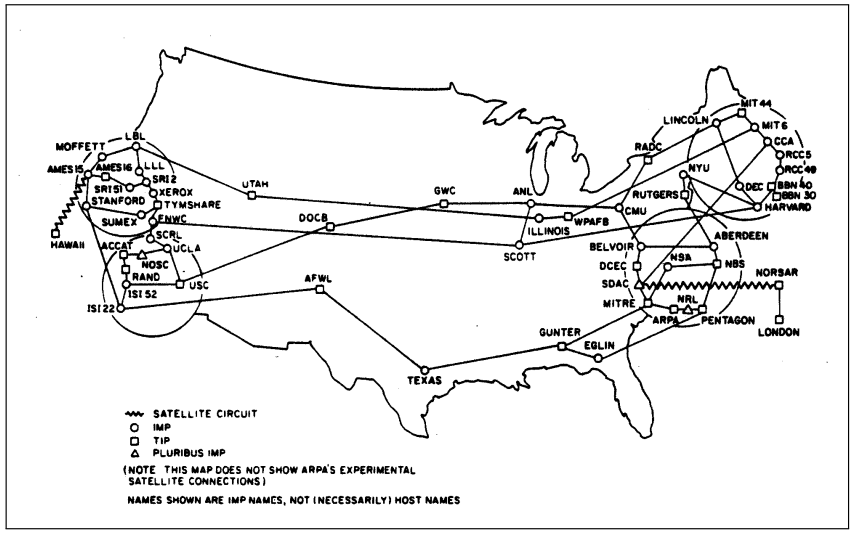
Example



Notes:

- You might prefer the original ASCII art from [9, Page 3]:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|       1       |       2       |       3       |       4       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|       5       |       6       |       7       |       8       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|       9       |       10      |       11      |       12      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

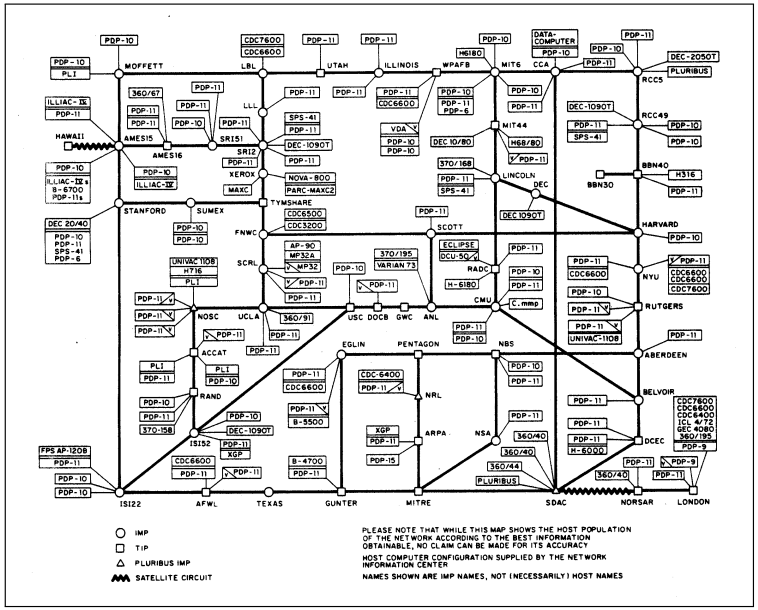



Notes:

- This diagram illustrates the physical connectivity present in ARPANET circa 1977. The majority of such connectivity was provided by dedicated, 56kbit s^{-1} leased telephone lines (provided by AT&T).

http://www.cs.utexas.edu/users/chris/DIGITAL_ARCHIVE/ARPANET/DARPA4779.pdf

An Aside: History

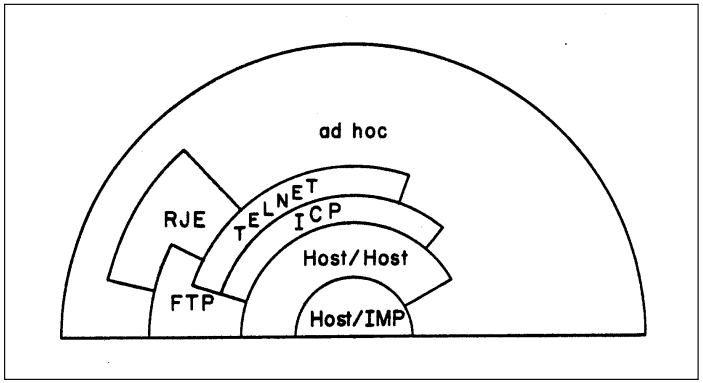


http://www.cs.utexas.edu/users/chris/DIGITAL_ARCHIVE/ARPANET/DARPA4779.pdf

Notes:

- This diagram illustrates the logical connectivity present in ARPANET circa 1977: notice that each site had multiple hosts, each linked to the ARPANET backbone using an IMP (which one can think of roughly as a router).

An Aside: History



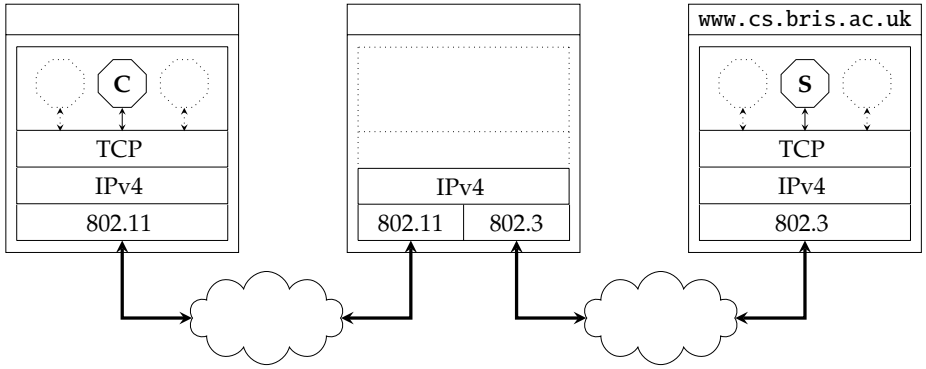
http://www.cs.utexas.edu/users/chris/DIGITAL_ARCHIVE/ARPANET/DARPA4779.pdf

Notes:

- This diagram illustrates (early) layered design wrt. the ARPANET protocol stack. Some of the terms and acronyms should be familiar (since they are still in use, e.g., FTP), others perhaps less so:
  - The **Remote Job Entry (RJE)** layer allows one host to initiate a task or process (or job, stemming from a batch processing context) on another. You can think of this as a (potentially higher-level) analogue to modern technologies such as **Remote Procedure Call (RPC)**.
  - An **Interface Message Processor (IMP)** was essentially a router in the original design: the idea was to have sub-networks at each site, with a dedicated IMP performing routing etc. Using such an approach a) reduced overhead on local resources, and b) managed the issue of inter-operation (since IMP to IMP communication could be uniform, without dependency on specificity in the sub-network).

Conclusions

- As a final step, we can add some detail to our running example, e.g.,



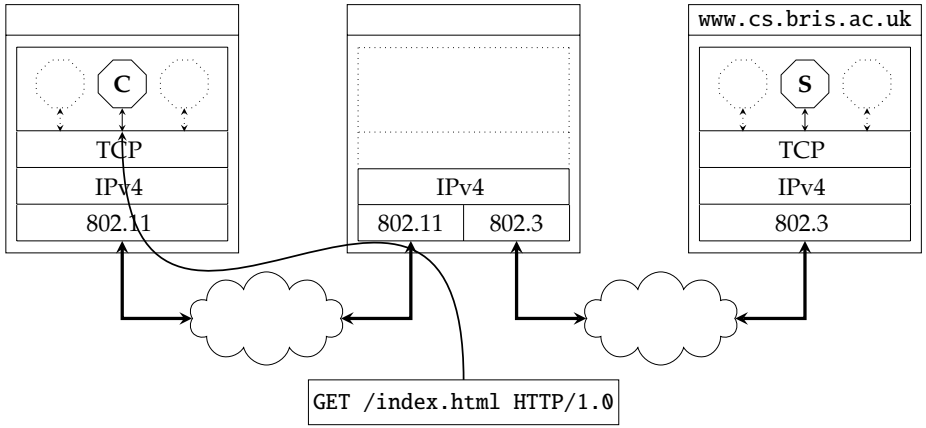
with the rest of the unit aiming to further explain each layer.

Notes:

- The router, which is in the center of the diagram, has *two* boxes at the link layer. This *should* make some sense, in that the router is connected to two networks: the two boxes illustrate the fact it has two network **interfaces**. Note that in this case the interfaces are for two different network types (namely 802.11 on the left, and 802.3 on the right) but the router would need two interfaces *even if* the types were the same.
- There are a few (perhaps subtle) points to take away from this diagram:
  1. We can cater for multiple processes on each host, since the transport layer will have access to a (de)multiplexing key (i.e., a port number) and can hence communicate data to and from the right one.
  2. We can cater for multiple compatible protocol choices (e.g., a 802.11 network on the left and 802.3 on the right), and indeed the networks may simultaneously be communicating data relating to *different* protocols (e.g., TCP *and* UDP both encapsulated in IPv4 packets and then in 802.11 frames); the latter also implies a need for IPv4 to have access to the right (de)multiplexing key (i.e., a protocol type) so it can communicate data to and from the right protocol in the transport layer.
  3. We must, and can cater for multiple access to the communication medium since there may be numerous hosts connected to each network.
  4. Although there is one router shown here, connected to two networks, it is possible to construct an increasingly complex inter-network via the same basic strategy: the strategy is scalable.

Conclusions

- As a final step, we can add some detail to our running example, e.g.,



with the rest of the unit aiming to further explain each layer.

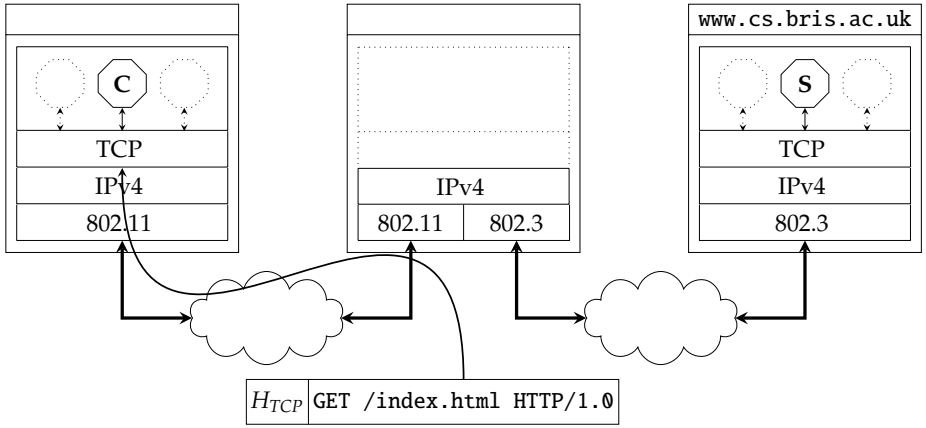
Daniel Page ⟨daniel.page@bristol.ac.uk⟩
Concurrent Computing (Computer Networks) git # 2a1cd85 @ 2016-03-05 University of BRISTOL

Notes:

- The router, which is in the center of the diagram, has *two* boxes at the link layer. This *should* make some sense, in that the router is connected to two networks: the two boxes illustrate the fact it has two network **interfaces**. Note that in this case the interfaces are for two different network types (namely 802.11 on the left, and 802.3 on the right) but the router would need two interfaces *even if* the types were the same.
- There are a few (perhaps subtle) points to take away from this diagram:
  1. We can cater for multiple processes on each host, since the transport layer will have access to a (de)multiplexing key (i.e., a port number) and can hence communicate data to and from the right one.
  2. We can cater for multiple compatible protocol choices (e.g., a 802.11 network on the left and 802.3 on the right), and indeed the networks may simultaneously be communicating data relating to *different* protocols (e.g., TCP *and* UDP both encapsulated in IPv4 packets and then in 802.11 frames); the latter also implies a need for IPv4 to have access to the right (de)multiplexing key (i.e., a protocol type) so it can communicate data to and from the right protocol in the transport layer.
  3. We must, and can cater for multiple access to the communication medium since there may be numerous hosts connected to each network.
  4. Although there is one router shown here, connected to two networks, it is possible to construct an increasingly complex inter-network via the same basic strategy: the strategy is scalable.

Conclusions

- As a final step, we can add some detail to our running example, e.g.,



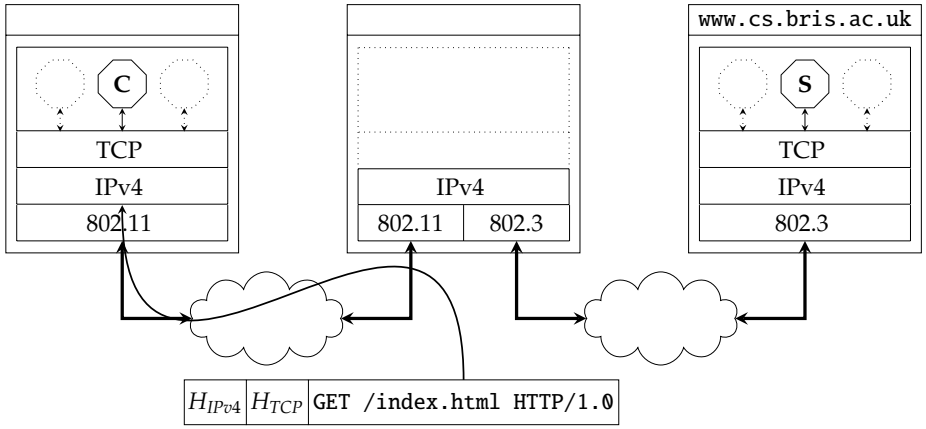
with the rest of the unit aiming to further explain each layer.

Notes:

- The router, which is in the center of the diagram, has *two* boxes at the link layer. This *should* make some sense, in that the router is connected to two networks: the two boxes illustrate the fact it has two network **interfaces**. Note that in this case the interfaces are for two different network types (namely 802.11 on the left, and 802.3 on the right) but the router would need two interfaces *even if* the types were the same.
- There are a few (perhaps subtle) points to take away from this diagram:
  1. We can cater for multiple processes on each host, since the transport layer will have access to a (de)multiplexing key (i.e., a port number) and can hence communicate data to and from the right one.
  2. We can cater for multiple compatible protocol choices (e.g., a 802.11 network on the left and 802.3 on the right), and indeed the networks may simultaneously be communicating data relating to *different* protocols (e.g., TCP *and* UDP both encapsulated in IPv4 packets and then in 802.11 frames); the latter also implies a need for IPv4 to have access to the right (de)multiplexing key (i.e., a protocol type) so it can communicate data to and from the right protocol in the transport layer.
  3. We must, and can cater for multiple access to the communication medium since there may be numerous hosts connected to each network.
  4. Although there is one router shown here, connected to two networks, it is possible to construct an increasingly complex inter-network via the same basic strategy: the strategy is scalable.

Conclusions

- As a final step, we can add some detail to our running example, e.g.,



with the rest of the unit aiming to further explain each layer.

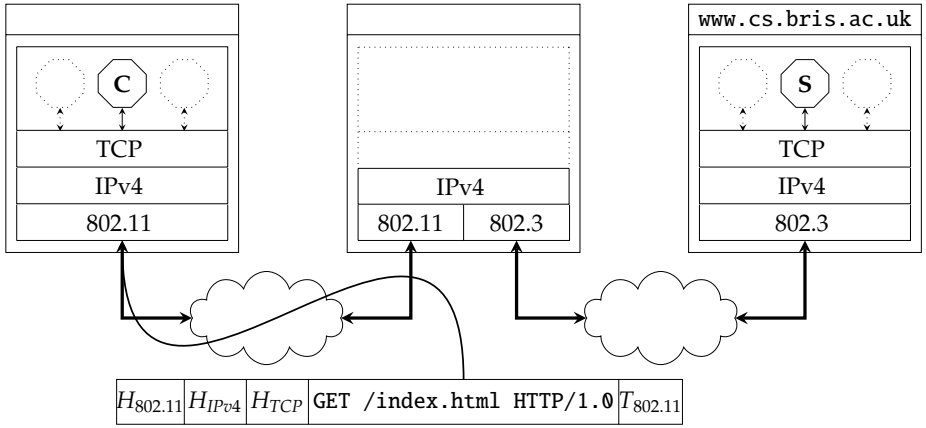
Daniel Page ⟨daniel.page@bristol.ac.uk⟩
Concurrent Computing (Computer Networks) git # 2a1cd85 @ 2016-03-05 University of BRISTOL

Notes:

- The router, which is in the center of the diagram, has *two* boxes at the link layer. This *should* make some sense, in that the router is connected to two networks: the two boxes illustrate the fact it has two network **interfaces**. Note that in this case the interfaces are for two different network types (namely 802.11 on the left, and 802.3 on the right) but the router would need two interfaces *even if* the types were the same.
- There are a few (perhaps subtle) points to take away from this diagram:
  1. We can cater for multiple processes on each host, since the transport layer will have access to a (de)multiplexing key (i.e., a port number) and can hence communicate data to and from the right one.
  2. We can cater for multiple compatible protocol choices (e.g., a 802.11 network on the left and 802.3 on the right), and indeed the networks may simultaneously be communicating data relating to *different* protocols (e.g., TCP *and* UDP both encapsulated in IPv4 packets and then in 802.11 frames); the latter also implies a need for IPv4 to have access to the right (de)multiplexing key (i.e., a protocol type) so it can communicate data to and from the right protocol in the transport layer.
  3. We must, and can cater for multiple access to the communication medium since there may be numerous hosts connected to each network.
  4. Although there is one router shown here, connected to two networks, it is possible to construct an increasingly complex inter-network via the same basic strategy: the strategy is scalable.

Conclusions

- As a final step, we can add some detail to our running example, e.g.,



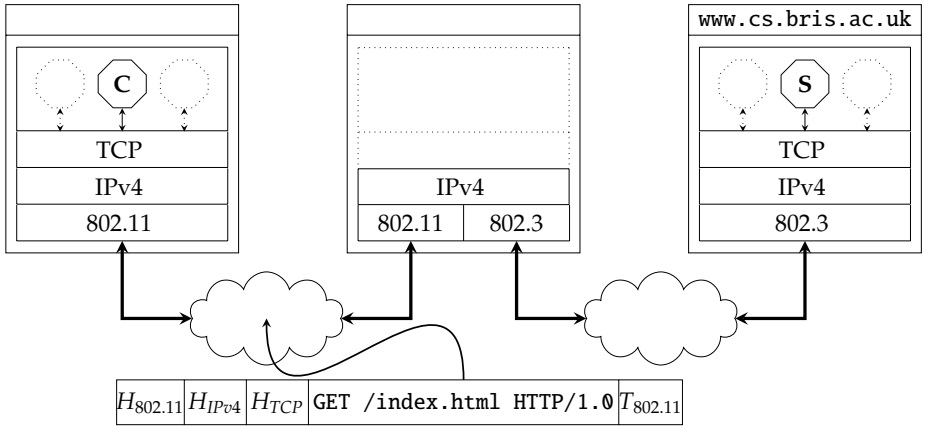
with the rest of the unit aiming to further explain each layer.

Notes:

- The router, which is in the center of the diagram, has *two* boxes at the link layer. This *should* make some sense, in that the router is connected to two networks: the two boxes illustrate the fact it has two network **interfaces**. Note that in this case the interfaces are for two different network types (namely 802.11 on the left, and 802.3 on the right) but the router would need two interfaces *even if* the types were the same.
- There are a few (perhaps subtle) points to take away from this diagram:
  1. We can cater for multiple processes on each host, since the transport layer will have access to a (de)multiplexing key (i.e., a port number) and can hence communicate data to and from the right one.
  2. We can cater for multiple compatible protocol choices (e.g., a 802.11 network on the left and 802.3 on the right), and indeed the networks may simultaneously be communicating data relating to *different* protocols (e.g., TCP *and* UDP both encapsulated in IPv4 packets and then in 802.11 frames); the latter also implies a need for IPv4 to have access to the right (de)multiplexing key (i.e., a protocol type) so it can communicate data to and from the right protocol in the transport layer.
  3. We must, and can cater for multiple access to the communication medium since there may be numerous hosts connected to each network.
  4. Although there is one router shown here, connected to two networks, it is possible to construct an increasingly complex inter-network via the same basic strategy: the strategy is scalable.

Conclusions

- As a final step, we can add some detail to our running example, e.g.,



with the rest of the unit aiming to further explain each layer.

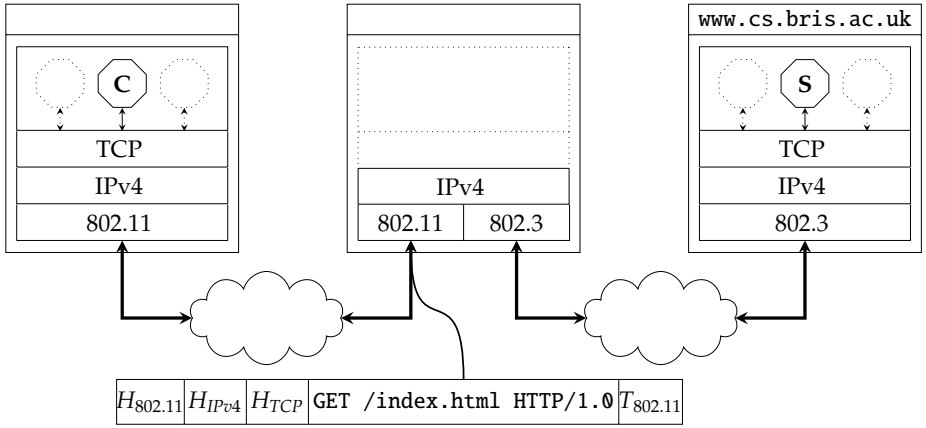
Daniel Page ⟨daniel.page@bristol.ac.uk⟩
Concurrent Computing (Computer Networks) git # 2a1cd85 @ 2016-03-05 University of BRISTOL

Notes:

- The router, which is in the center of the diagram, has *two* boxes at the link layer. This *should* make some sense, in that the router is connected to two networks: the two boxes illustrate the fact it has two network **interfaces**. Note that in this case the interfaces are for two different network types (namely 802.11 on the left, and 802.3 on the right) but the router would need two interfaces *even if* the types were the same.
- There are a few (perhaps subtle) points to take away from this diagram:
  1. We can cater for multiple processes on each host, since the transport layer will have access to a (de)multiplexing key (i.e., a port number) and can hence communicate data to and from the right one.
  2. We can cater for multiple compatible protocol choices (e.g., a 802.11 network on the left and 802.3 on the right), and indeed the networks may simultaneously be communicating data relating to *different* protocols (e.g., TCP *and* UDP both encapsulated in IPv4 packets and then in 802.11 frames); the latter also implies a need for IPv4 to have access to the right (de)multiplexing key (i.e., a protocol type) so it can communicate data to and from the right protocol in the transport layer.
  3. We must, and can cater for multiple access to the communication medium since there may be numerous hosts connected to each network.
  4. Although there is one router shown here, connected to two networks, it is possible to construct an increasingly complex inter-network via the same basic strategy: the strategy is scalable.

Conclusions

- As a final step, we can add some detail to our running example, e.g.,



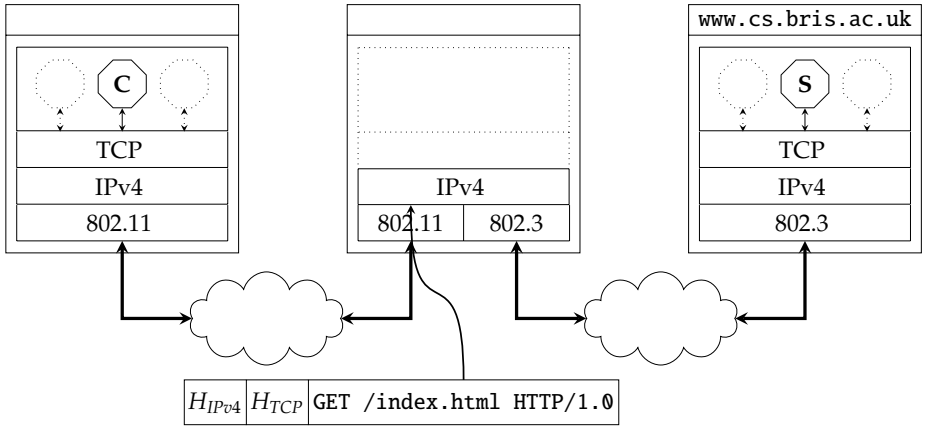
with the rest of the unit aiming to further explain each layer.

Notes:

- The router, which is in the center of the diagram, has *two* boxes at the link layer. This *should* make some sense, in that the router is connected to two networks: the two boxes illustrate the fact it has two network **interfaces**. Note that in this case the interfaces are for two different network types (namely 802.11 on the left, and 802.3 on the right) but the router would need two interfaces *even if* the types were the same.
- There are a few (perhaps subtle) points to take away from this diagram:
  1. We can cater for multiple processes on each host, since the transport layer will have access to a (de)multiplexing key (i.e., a port number) and can hence communicate data to and from the right one.
  2. We can cater for multiple compatible protocol choices (e.g., a 802.11 network on the left and 802.3 on the right), and indeed the networks may simultaneously be communicating data relating to *different* protocols (e.g., TCP *and* UDP both encapsulated in IPv4 packets and then in 802.11 frames); the latter also implies a need for IPv4 to have access to the right (de)multiplexing key (i.e., a protocol type) so it can communicate data to and from the right protocol in the transport layer.
  3. We must, and can cater for multiple access to the communication medium since there may be numerous hosts connected to each network.
  4. Although there is one router shown here, connected to two networks, it is possible to construct an increasingly complex inter-network via the same basic strategy: the strategy is scalable.

Conclusions

- As a final step, we can add some detail to our running example, e.g.,



with the rest of the unit aiming to further explain each layer.

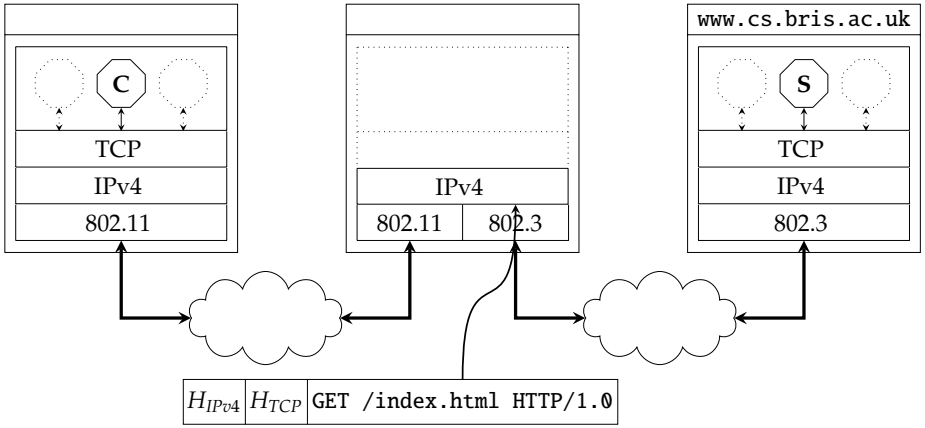
Daniel Page ⟨daniel.page@bristol.ac.uk⟩
Concurrent Computing (Computer Networks) git # 2a1cd85 @ 2016-03-05 University of BRISTOL

Notes:

- The router, which is in the center of the diagram, has *two* boxes at the link layer. This *should* make some sense, in that the router is connected to two networks: the two boxes illustrate the fact it has two network **interfaces**. Note that in this case the interfaces are for two different network types (namely 802.11 on the left, and 802.3 on the right) but the router would need two interfaces *even if* the types were the same.
- There are a few (perhaps subtle) points to take away from this diagram:
  1. We can cater for multiple processes on each host, since the transport layer will have access to a (de)multiplexing key (i.e., a port number) and can hence communicate data to and from the right one.
  2. We can cater for multiple compatible protocol choices (e.g., a 802.11 network on the left and 802.3 on the right), and indeed the networks may simultaneously be communicating data relating to *different* protocols (e.g., TCP *and* UDP both encapsulated in IPv4 packets and then in 802.11 frames); the latter also implies a need for IPv4 to have access to the right (de)multiplexing key (i.e., a protocol type) so it can communicate data to and from the right protocol in the transport layer.
  3. We must, and can cater for multiple access to the communication medium since there may be numerous hosts connected to each network.
  4. Although there is one router shown here, connected to two networks, it is possible to construct an increasingly complex inter-network via the same basic strategy: the strategy is scalable.

Conclusions

- As a final step, we can add some detail to our running example, e.g.,



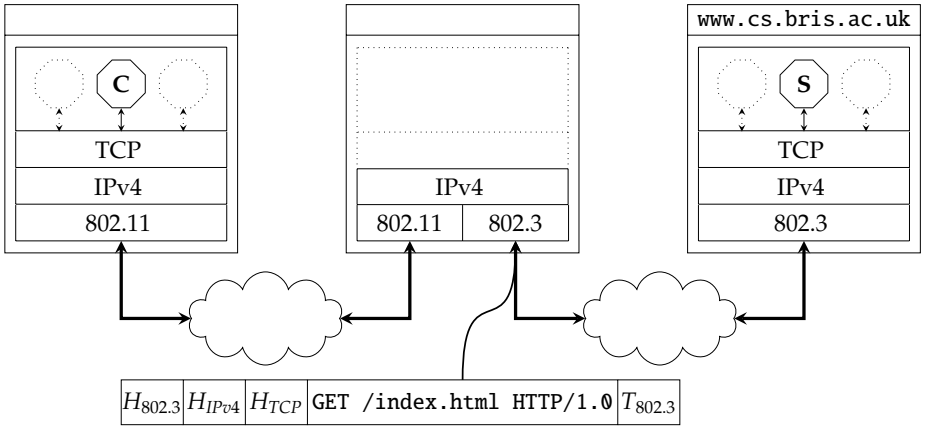
with the rest of the unit aiming to further explain each layer.

Notes:

- The router, which is in the center of the diagram, has *two* boxes at the link layer. This *should* make some sense, in that the router is connected to two networks: the two boxes illustrate the fact it has two network **interfaces**. Note that in this case the interfaces are for two different network types (namely 802.11 on the left, and 802.3 on the right) but the router would need two interfaces *even if* the types were the same.
- There are a few (perhaps subtle) points to take away from this diagram:
  1. We can cater for multiple processes on each host, since the transport layer will have access to a (de)multiplexing key (i.e., a port number) and can hence communicate data to and from the right one.
  2. We can cater for multiple compatible protocol choices (e.g., a 802.11 network on the left and 802.3 on the right), and indeed the networks may simultaneously be communicating data relating to *different* protocols (e.g., TCP *and* UDP both encapsulated in IPv4 packets and then in 802.11 frames); the latter also implies a need for IPv4 to have access to the right (de)multiplexing key (i.e., a protocol type) so it can communicate data to and from the right protocol in the transport layer.
  3. We must, and can cater for multiple access to the communication medium since there may be numerous hosts connected to each network.
  4. Although there is one router shown here, connected to two networks, it is possible to construct an increasingly complex inter-network via the same basic strategy: the strategy is scalable.

Conclusions

- As a final step, we can add some detail to our running example, e.g.,



with the rest of the unit aiming to further explain each layer.

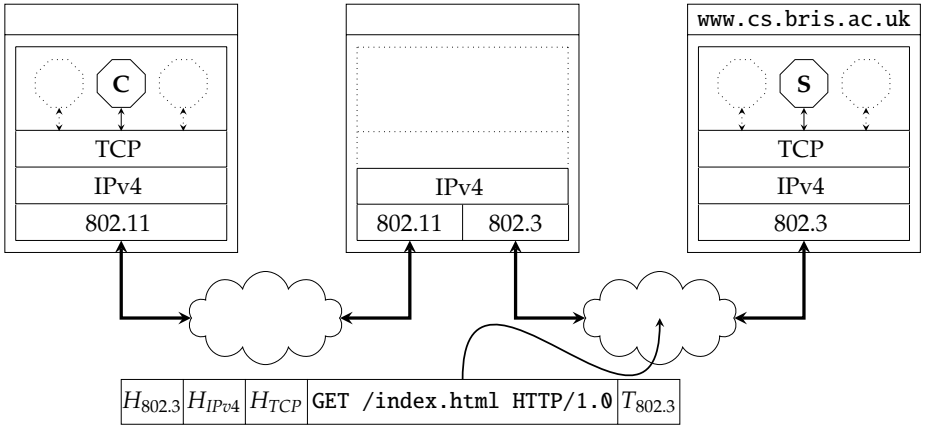
Daniel Page ⟨daniel.page@bristol.ac.uk⟩
Concurrent Computing (Computer Networks) git # 2a1cd85 @ 2016-03-05 University of BRISTOL

Notes:

- The router, which is in the center of the diagram, has *two* boxes at the link layer. This *should* make some sense, in that the router is connected to two networks: the two boxes illustrate the fact it has two network **interfaces**. Note that in this case the interfaces are for two different network types (namely 802.11 on the left, and 802.3 on the right) but the router would need two interfaces *even if* the types were the same.
- There are a few (perhaps subtle) points to take away from this diagram:
  1. We can cater for multiple processes on each host, since the transport layer will have access to a (de)multiplexing key (i.e., a port number) and can hence communicate data to and from the right one.
  2. We can cater for multiple compatible protocol choices (e.g., a 802.11 network on the left and 802.3 on the right), and indeed the networks may simultaneously be communicating data relating to *different* protocols (e.g., TCP *and* UDP both encapsulated in IPv4 packets and then in 802.11 frames); the latter also implies a need for IPv4 to have access to the right (de)multiplexing key (i.e., a protocol type) so it can communicate data to and from the right protocol in the transport layer.
  3. We must, and can cater for multiple access to the communication medium since there may be numerous hosts connected to each network.
  4. Although there is one router shown here, connected to two networks, it is possible to construct an increasingly complex inter-network via the same basic strategy: the strategy is scalable.

Conclusions

- As a final step, we can add some detail to our running example, e.g.,



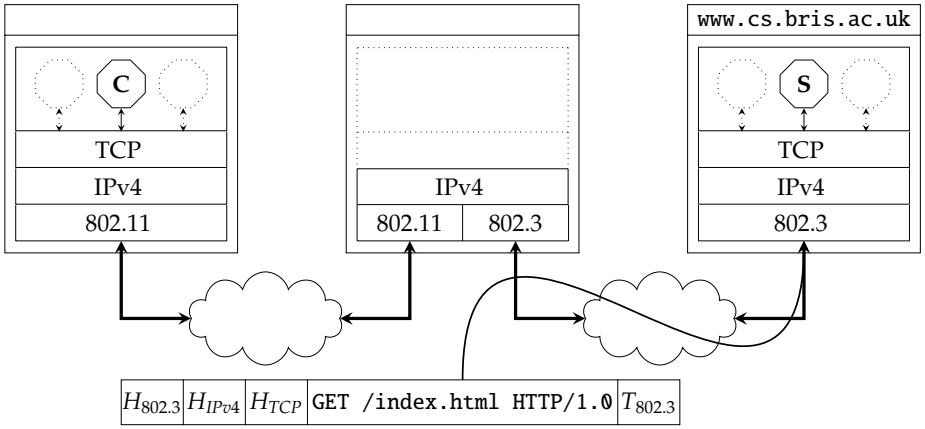
with the rest of the unit aiming to further explain each layer.

Notes:

- The router, which is in the center of the diagram, has *two* boxes at the link layer. This *should* make some sense, in that the router is connected to two networks: the two boxes illustrate the fact it has two network **interfaces**. Note that in this case the interfaces are for two different network types (namely 802.11 on the left, and 802.3 on the right) but the router would need two interfaces *even if* the types were the same.
- There are a few (perhaps subtle) points to take away from this diagram:
  1. We can cater for multiple processes on each host, since the transport layer will have access to a (de)multiplexing key (i.e., a port number) and can hence communicate data to and from the right one.
  2. We can cater for multiple compatible protocol choices (e.g., a 802.11 network on the left and 802.3 on the right), and indeed the networks may simultaneously be communicating data relating to *different* protocols (e.g., TCP *and* UDP both encapsulated in IPv4 packets and then in 802.11 frames); the latter also implies a need for IPv4 to have access to the right (de)multiplexing key (i.e., a protocol type) so it can communicate data to and from the right protocol in the transport layer.
  3. We must, and can cater for multiple access to the communication medium since there may be numerous hosts connected to each network.
  4. Although there is one router shown here, connected to two networks, it is possible to construct an increasingly complex inter-network via the same basic strategy: the strategy is scalable.

Conclusions

- As a final step, we can add some detail to our running example, e.g.,



with the rest of the unit aiming to further explain each layer.

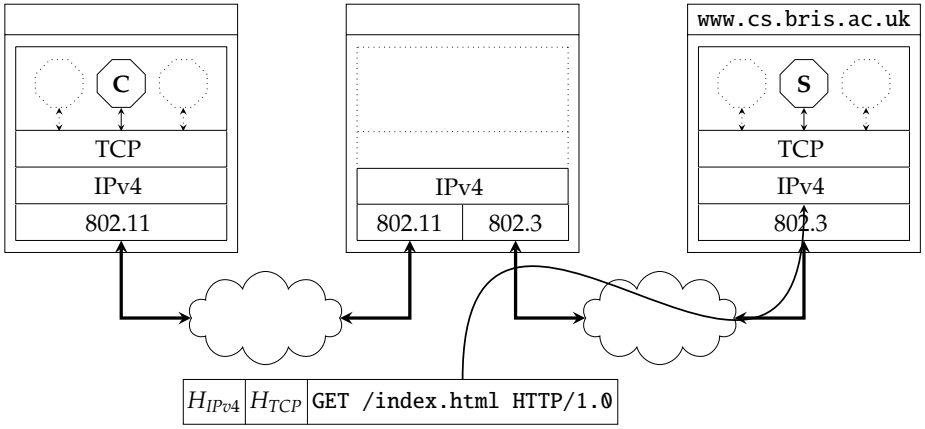
Daniel Page ⟨daniel.page@bristol.ac.uk⟩
Concurrent Computing (Computer Networks) git # 2a1cd85 @ 2016-03-05 University of BRISTOL

Notes:

- The router, which is in the center of the diagram, has *two* boxes at the link layer. This *should* make some sense, in that the router is connected to two networks: the two boxes illustrate the fact it has two network **interfaces**. Note that in this case the interfaces are for two different network types (namely 802.11 on the left, and 802.3 on the right) but the router would need two interfaces *even if* the types were the same.
- There are a few (perhaps subtle) points to take away from this diagram:
  1. We can cater for multiple processes on each host, since the transport layer will have access to a (de)multiplexing key (i.e., a port number) and can hence communicate data to and from the right one.
  2. We can cater for multiple compatible protocol choices (e.g., a 802.11 network on the left and 802.3 on the right), and indeed the networks may simultaneously be communicating data relating to *different* protocols (e.g., TCP *and* UDP both encapsulated in IPv4 packets and then in 802.11 frames); the latter also implies a need for IPv4 to have access to the right (de)multiplexing key (i.e., a protocol type) so it can communicate data to and from the right protocol in the transport layer.
  3. We must, and can cater for multiple access to the communication medium since there may be numerous hosts connected to each network.
  4. Although there is one router shown here, connected to two networks, it is possible to construct an increasingly complex inter-network via the same basic strategy: the strategy is scalable.

Conclusions

- As a final step, we can add some detail to our running example, e.g.,



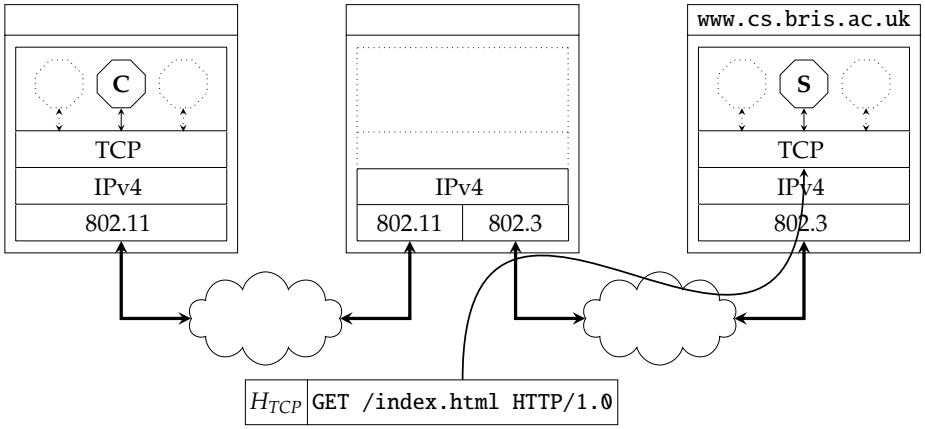
with the rest of the unit aiming to further explain each layer.

Notes:

- The router, which is in the center of the diagram, has *two* boxes at the link layer. This *should* make some sense, in that the router is connected to two networks: the two boxes illustrate the fact it has two network **interfaces**. Note that in this case the interfaces are for two different network types (namely 802.11 on the left, and 802.3 on the right) but the router would need two interfaces *even if* the types were the same.
- There are a few (perhaps subtle) points to take away from this diagram:
  1. We can cater for multiple processes on each host, since the transport layer will have access to a (de)multiplexing key (i.e., a port number) and can hence communicate data to and from the right one.
  2. We can cater for multiple compatible protocol choices (e.g., a 802.11 network on the left and 802.3 on the right), and indeed the networks may simultaneously be communicating data relating to *different* protocols (e.g., TCP *and* UDP both encapsulated in IPv4 packets and then in 802.11 frames); the latter also implies a need for IPv4 to have access to the right (de)multiplexing key (i.e., a protocol type) so it can communicate data to and from the right protocol in the transport layer.
  3. We must, and can cater for multiple access to the communication medium since there may be numerous hosts connected to each network.
  4. Although there is one router shown here, connected to two networks, it is possible to construct an increasingly complex inter-network via the same basic strategy: the strategy is scalable.

Conclusions

- As a final step, we can add some detail to our running example, e.g.,



with the rest of the unit aiming to further explain each layer.

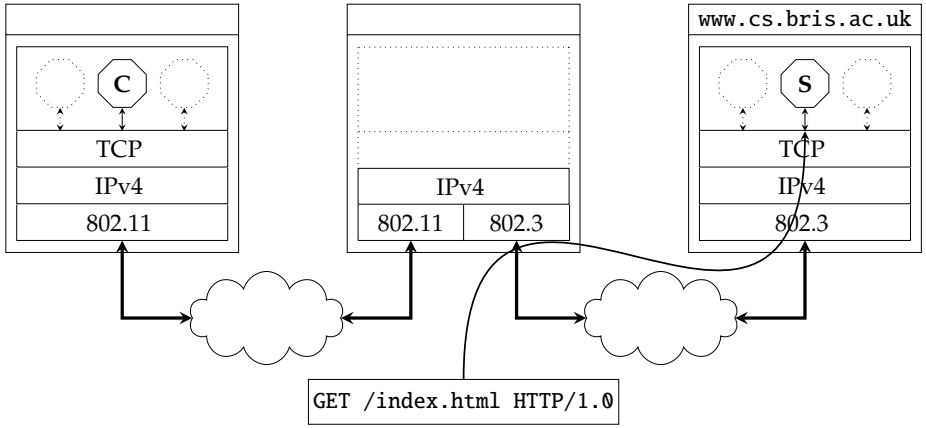
Daniel Page ⟨daniel.page@bristol.ac.uk⟩
Concurrent Computing (Computer Networks) git # 2a1cd85 @ 2016-03-05 University of BRISTOL

Notes:

- The router, which is in the center of the diagram, has *two* boxes at the link layer. This *should* make some sense, in that the router is connected to two networks: the two boxes illustrate the fact it has two network **interfaces**. Note that in this case the interfaces are for two different network types (namely 802.11 on the left, and 802.3 on the right) but the router would need two interfaces *even if* the types were the same.
- There are a few (perhaps subtle) points to take away from this diagram:
  1. We can cater for multiple processes on each host, since the transport layer will have access to a (de)multiplexing key (i.e., a port number) and can hence communicate data to and from the right one.
  2. We can cater for multiple compatible protocol choices (e.g., a 802.11 network on the left and 802.3 on the right), and indeed the networks may simultaneously be communicating data relating to *different* protocols (e.g., TCP *and* UDP both encapsulated in IPv4 packets and then in 802.11 frames); the latter also implies a need for IPv4 to have access to the right (de)multiplexing key (i.e., a protocol type) so it can communicate data to and from the right protocol in the transport layer.
  3. We must, and can cater for multiple access to the communication medium since there may be numerous hosts connected to each network.
  4. Although there is one router shown here, connected to two networks, it is possible to construct an increasingly complex inter-network via the same basic strategy: the strategy is scalable.

Conclusions

- As a final step, we can add some detail to our running example, e.g.,



with the rest of the unit aiming to further explain each layer.

Notes:

- The router, which is in the center of the diagram, has *two* boxes at the link layer. This *should* make some sense, in that the router is connected to two networks: the two boxes illustrate the fact it has two network **interfaces**. Note that in this case the interfaces are for two different network types (namely 802.11 on the left, and 802.3 on the right) but the router would need two interfaces *even if* the types were the same.
- There are a few (perhaps subtle) points to take away from this diagram:
  1. We can cater for multiple processes on each host, since the transport layer will have access to a (de)multiplexing key (i.e., a port number) and can hence communicate data to and from the right one.
  2. We can cater for multiple compatible protocol choices (e.g., a 802.11 network on the left and 802.3 on the right), and indeed the networks may simultaneously be communicating data relating to *different* protocols (e.g., TCP *and* UDP both encapsulated in IPv4 packets and then in 802.11 frames); the latter also implies a need for IPv4 to have access to the right (de)multiplexing key (i.e., a protocol type) so it can communicate data to and from the right protocol in the transport layer.
  3. We must, and can cater for multiple access to the communication medium since there may be numerous hosts connected to each network.
  4. Although there is one router shown here, connected to two networks, it is possible to construct an increasingly complex inter-network via the same basic strategy: the strategy is scalable.

Conclusions

- Take away points:
  - Although a lot of this content might seem very abstract, the point is that we now have a sensible high-level design.
  - We'll fill in missing detail using a bottom-up approach by covering the
    1. link layer,
    2. internet layer,
    3. transport layer, and
    4. application layer

    while emphasising the *general* concepts as applied in specific technologies ...
  - ... keeping in mind the running, motivating example of HTTP.
  - A central challenge throughout is design of solutions that work efficiently in the general-case, but guarentee correctness in (many) special- or corner-cases.

Notes:

References

[1] Wikipedia: Circuit switching.
http://en.wikipedia.org/wiki/Circuit_switching.

[2] Wikipedia: Internet protocol suite.
http://en.wikipedia.org/wiki/Internet_protocol_suite.

[3] Wikipedia: Network topology.
http://en.wikipedia.org/wiki/Network_topology.

[4] Wikipedia: OSI model.
http://en.wikipedia.org/wiki/OSI_model.

[5] Wikipedia: Packet switching.
http://en.wikipedia.org/wiki/Packet_switching.

[6] Wikipedia: Protocol stack.
http://en.wikipedia.org/wiki/Protocol_stack.

[7] D. Cohen.
On holy wars and a plea for peace.
IEEE Computer Magazine, 14:48–54, 1981.

[8] J. Postel.
Transmision Control Protocol.
Internet Engineering Task Force (IETF) Request for Comments (RFC) 793, 1981.
http://tools.ietf.org/html/rfc793.

Notes:

References

[9] J. Reynolds and J. Postel.
Assigned numbers.
Internet Engineering Task Force (IETF) Request for Comments (RFC) 1060, 1990.
http://tools.ietf.org/html/rfc1060.

[10] W. Stallings.
Chapter 11: Circuit switching and packet switching.
In *Data and Computer Communications* [13].

[11] W. Stallings.
Chapter 2: Data communications, data networks and the Internet.
In *Data and Computer Communications* [13].

[12] W. Stallings.
Chapter 3: Protocol architecture, TCP/IP, and Internet-based applications.
In *Data and Computer Communications* [13].

[13] W. Stallings.
Data and Computer Communications.
Pearson, 9th edition, 2010.

Notes: