## Databases

Data Model: A collection of intuitive concepts describing data, their relationships and their constraints.

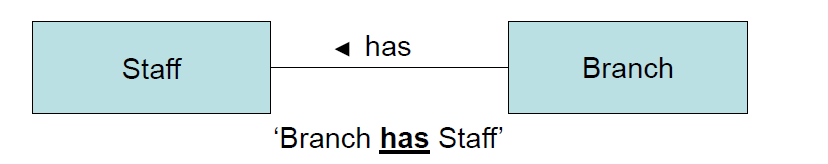
Entity type: groups of objects with the same properties, identified as having an independent existence.

Entity occurrence: uniquely identifiable instance of an entity type.

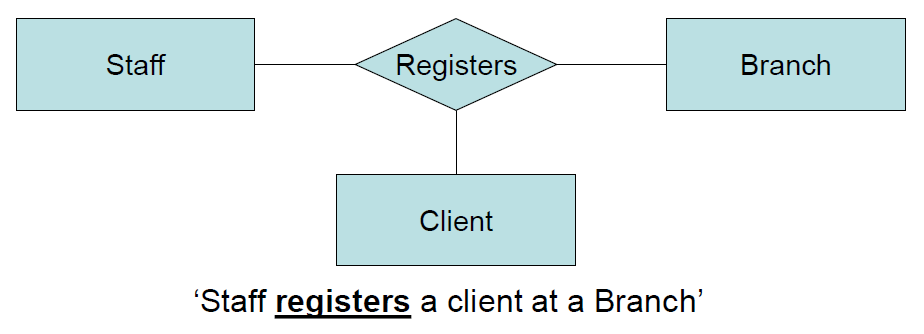
The diagrammatic representation of entites is a labelled rectangle with the entity name.

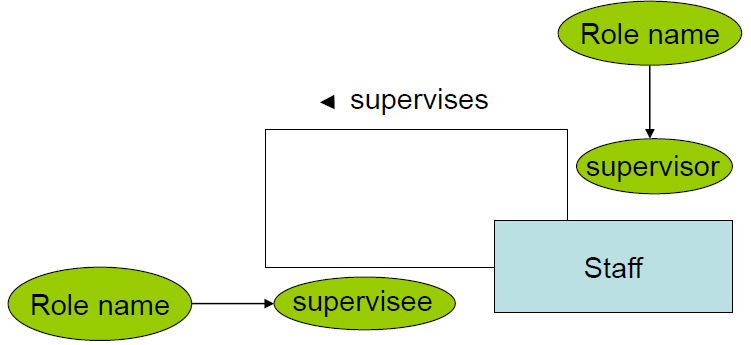
Relationship type: a meaningful association between two or more entity types.

Degree of a relationship: the number of entities that participatein a relationship.

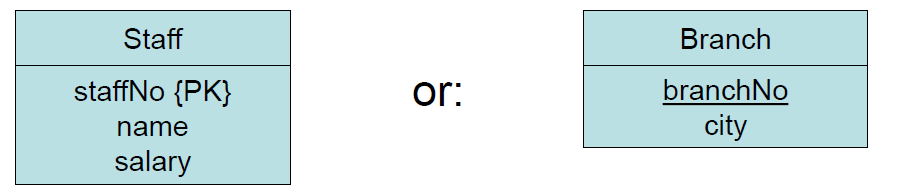
Diagrammatic representation of a binary relationship: has a direction

Diagrammatic representation of an n-ary relationship (n>2): doesn’t necessarily have a direction

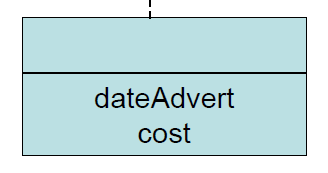


When n = 1 unary or recursive relationship. Role names are required to indicate the purpose of each entity occurrence in the relationship.Role names are used at the beginning and end of the line. They can also be used when entities are associated through more than one relationship.

Factors for choosing a primary key: number of attributes, length of an attribute (smaller = better), future certainty of uniqueness



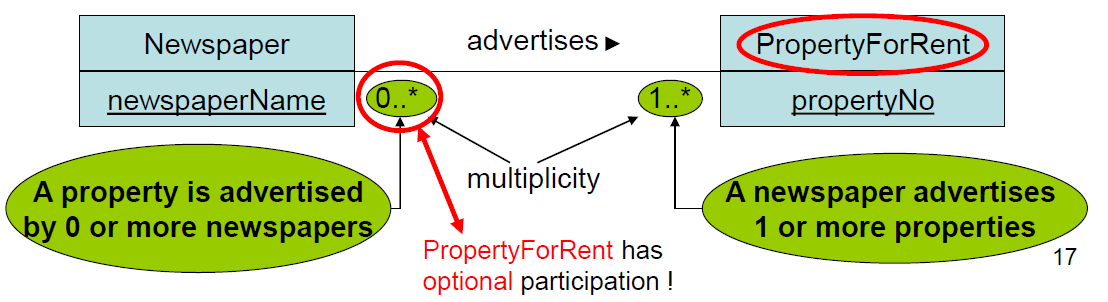
Diagrammatic representation of an entity



Diagrammatic representation of a relationship. Connected by a dashed line to the line connecting the entities.

Multiplicity of a relationship: number of entity occurrences to which another entity can be associated with this relationship.

Diagrammatic representation: the minimum and maximum number of occurrences of each entity is written. The multiplicity is written on the side opposite to the entity.



For an n-ary relationship, the number of entity occurrences when the n-1 occurrences are fixed for the other entity types.

The basic concepts of the ER modelling are limiting when modelling modern and complex DB applications with large amounts of data. The solution to this is additonal semantic modeling concepts and to reduce the complexity of the ER model to make it more intuitive. This leads to the

### Enhanced Entity relationship model (EER)

The main concepts are specialization and generalization.

Subclass: subgrouping of occurrences of an entity type which requires it to be represented separately eg manager, supervisor

Subclass: entity type that has two or more distinct subclasses. Eg staff.

Superclass/ subclass relationship: each member of a subclass is also a member of a superclass.

Attribute inheritence: All attributes of a superclass are also attributes of a subclass. A subclass can have additional attributes (like in OOP).

Advantages:

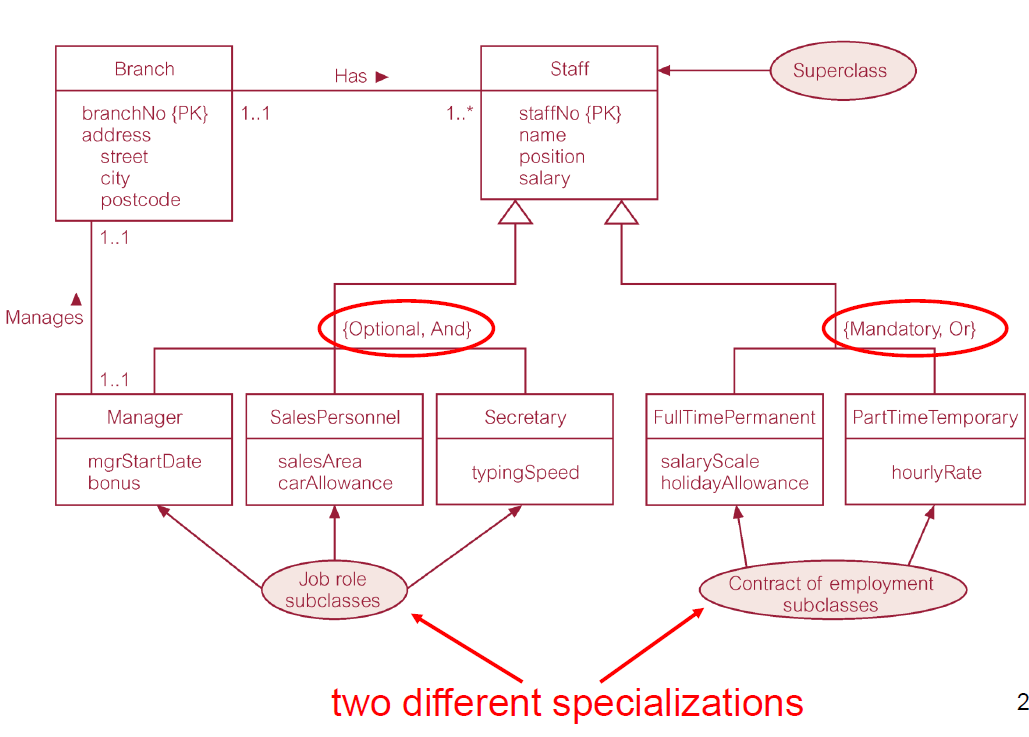
* Avoid describing similar concepts more than once
* Have relations that include a subclass but not the superclass
* More semantic information to the design.

Type hierarchy: an entity with its subclasses and their subclasses etc. aka specialization hierarchy (manager is a specialization of staff), generalization hierarchy (staff is a generalization of manager), IS-A hierarchy (manager IS-A member of staff).

Specialization: top down approach process of maximizing the differences between entity occurrences by identifying their distinguishing characteristics.

Generalization: bottom up process of minimizing the differences between entity occurrences by identifying common characteristics.

Participation constraint determines whether every member in the superclass must particpate as a member of a subclass or not. Can be mandatory or optional.

Disjoint constraint determines whether a member of its superclass can be a member of more than one subclass. Only applies in case of 2 or more subclasses. Or is disjoint, can only belong to one subclass. And is non-disjoint, can be belong to more than one subclass.

Superclasses can have shared subclasses.

The arrows show specialization/generalization. They point to the superclass.

There are 3 characterisations of data:

* Structured data: data represented in a strict format (eg schema). The DBMS checks to ensure the data follows the structures and integrity and referential constraints (primary/ foreign keysthat are specified in the schema.
* Semi-structured data: data that may be irregualar or incomplete and have a structure that may change rapidly or unpredictably. The data may have some structure but not all parts have the same fixed structure or every data objectmay have different attributes that are not known in advance. This is also known as self-describing or schema-less data. The schema information is mixed with the data values. A separate schema (or Document Type Definition) can exist, but this only places a loose contraint on data.
* Unstructured data: there is a very limited indication of the type/structure of data. Eg a web page in HTML that contains some data.

### Semi-structured data

The main language for semi-structured data is XML. XML describes the content of data (the semantics of the data). Tags such as <book> and </book> or <title> and </title>. Elements can be nested. An empty element is either <author></author> or <author/>. An XML document contains a single root element.

CASE SENSITIVE.

XML data has a tree structure, and so the ordering of elements matters.

XML is self-describing. Without a schema, only the relative position of elements in the tree matters. The schema is part of the data, and is discovered from the data. This makes XML flexible.

XML leads to easy data sharing and exchange on the web, similar to JSON.

Advantages of XML:

* Simple
* Extensibility: user defined tags
* Platform- and vendor-independent: works on all platforms and supports all alphabets.
* Separation of content and presentation

Different elements can have different structures.

An attrbibute is a name-value pair with descriptive / identifying information about an element, places inside the start tag of the element. Eg <STAFF branchNo=”B005”>. An attribute can only appear once in an element, but subelements can use the same attribute. Attributes in XML are not ordered.

Attributes can be replaced by sub-elements. Better practice to use sub-elements, unless attributes describe metadata.

Disadvantages of attributes:

* Cannot contain multiple values
* Cannot describe structure
* More difficult to manipulate by program code
* Not easily expandable
* Not easy to test against Document Type Definition

Some attributes can be declared as of type: ID (used to identify an element) and IDREF (used as a pointer to an element with a specific ID) and IDREFS (consists of a list of IDs, separated by a white space).

<!-- --> show comments. &…; shows special characters / reserved symbols. Eg &lt; = <

### Querying XML

XPath – basis of a template describing how to traverse a directed tree; navigates through the tree to a select path.

XQuery – the complete XML query language. Selects / combines data using XPath and constructs output.

XPath is a declarative (non-procedural) query language for XML.

* Simple syntax for addressing parts of an XML document
* Treats an XML document as a logical ordered tree with nodes.

Diagram

Description automatically generatedNavigation path: output is at the end of the path.

The nodes themselves are returned, not the text in the nodes.

Restricted Kleene closure: eg //author – find any node with the tag “author”.

@ matches an attributes

@\* matches any attribute

A predicate can be added after a tag – Boolean condition in square brackets eg //book[@price=“55”].

Comparisons have an implicit “There exists” sense. Eg /bib/book/author[firstName] matches elements having “firstName” as a sub-element. Can have multiple predicates, but the order they are listed in depends if they are independent or not.

General form of XPath query: /step1/step2/…/stepn. Each step consists of a basis (and optionally, a list of predicates). A basis consists of an axis (direction) and a node test (type of node to navigate to).

Axes include

* Child
* Parent
* Attribute
* Self
* Descendant
* Descendant-or-self
* Ancestor
* Ancestor-or-self
* Following-sibling
* Preceding-sibling
* Following
* Preceding

Following / preceding: nodes after/before the current node in the XML document order, which are not descendants or ancestors of the current node.

The syntax of using axes is:

Prefix a tag by an axis name and :: eg. /bib/book/@price === /child::bib/child::book/attribute::price

Text() matches text value

Name() returns the name of the current tag: /bib//book\*[name() = “book”] === /bib//book

A subelement of an elemet can be selected using predicate format: [position() = i] eg //author[position() = 2]

Alternatively select desired position: /bib/book/author[2].

### XQuery

XQuery selects / combines data (using XPath) and constructs an output. The result of a path expression is an ordered list of nodes, including their descendent nodes and may contain duplicates (multiple nodes with the same content).

The basic form of XQuery is similar to SQL:

FOR – variable bindings to individual nodes, iterates over many values, returns multiple variables (FOR $x IN doc(“bib.xml”)/bib/book)

LET – variable bindings to collections of nodes, returns a single value (LET $x := doc(“bib.xml”)/bib/book)

WHERE – qualifier conditions

RETURN – query result specification

Eg find all book titles published after 1995:

FOR $x in doc(“bib.xml”)/bib/book

WHERE $x/year > 1995

RETURN {$x/title}

The equivalent in XQuery is doc(“bib.xml”)/bib/book[year>1995]/title

/bib/book/author would return an ordered collection, distinct-values(/bib/book/author) returns an unordered collection. Distinct-values() removes duplicates from the result.

LET $a:=/bib/book - $a is a collection

$a/author is a collection of several authors

XML performs atomization (convert content of an element to a value) and casting (convert it to a value of appropriate type).

XQuery can be nested.

Aggregate functions:

* Count(), returns number of elements
* Avg() returns average
* Min()
* Max()
* Sum()

Other functions:

* Exists ()
* Empty ()

JOIN: used to join two XML documents. Give the docs different variables.

ORDER BY: after FOR

Existential quantifier: WHERE SOME $p IN whatever SATISFIES

Universal quantifier: WHERE EVERY $p IN whatever SATISFIES

### Transactions

Concurrency control protocols: prevent database accesses to intefere with each other

Database recovery: the process of restoring a database to a correct state after a failure

Transaction: an action or series of actions carried out by a single user or program which reads/ updates the database.

At the end of a transaction, the database is in a consistent state again: the referential and integrity contraints are valid. During the execution of a transaction, the database may be in an inconsistent state. A transaction can have one of two outcomes: commited (successful) or rolled back (unsuccesful). All transactions must have ACID properties:

* Atomicity:
  + “all or nothing” – the transaction is performed completely or not at all.
  + The recovery subsystem of the DBMS is responsible.
* Consistency:
  + Database must be transformed from a consistent state to another consistent state
  + The DBMS and application developers are responsible
* Isolation:
  + Transactions execute independently: effects of incomplete transactions are invisible to other transactions
  + Concurrency control system of the DBMS is responsible
* Durability
  + Effects of a committed transaction are permenantly recorded (in the disk). Never lost because of a failure.
  + Recovery subsystem of the DBMS is responsible.

Transaction manager: coordinates transactions on behalf of the application programs

Buffer manager: efficient transfer of data between disk and main memory

Recovery manager: in case of failure, restores the database to the last consistent state.

Scheduler: implements strategy for concurrency control. When locks are used, known as lock manager.

The aims of the scheduler are

1. Efficiency- maximise concurrency
2. Correctness – do not allow executing transactions to intefere with each other.

Concurrency control: process of managing simultaneous operations on the database, without having them intefere with each other.

3 types of problems with intefering transactions:

* Lost update problem
  + “override by mistake”
  + An apparently successful completed update operation by one user is overriden by another.
* Uncommitted dependency problem
  + A transaction is allowed to see the intermediate results of a transaction (“dirty data”) before it is committed.
  + Can be prevented by preventing transactions from reading data until another commits or aborts.
* Inconsistent analysis
  + Transaction reads some values while they are being updated by another

The obvious solution to all of these is to only allow one transaction at a time. However, we want to maximise concurrency (parallelism).

Schedule: sequence of operaations from a set of n concurrent transactions such that the order of operations in each transaction is preserved in the schedule.

Serial schedule: schedule where operations of some transaction are not interleaved – the order of transactions matter. One transaction finishes and then the next one starts.

Serializable schedules – any serial schedule leaves the database in a consistent state. Different schedules result in different states. A non serial schedule is serializable (correct) if it produces a database state that can be produced by some serial execution of the same transactions.

Find an equivalent serial schedule:

* Non conflicting pairs of operations – 2 transactions only read a data item and the two transactions read or write completely separate data items.
* Conflicting pairs of operations - one transaction writes a data item and another reads or writes the same data item

In serializability, ordering only matters for conflicting transactions. Construct a precedence graph to check whether a non-serial schedule is serializable or not.

Diagram

Description automatically generatedText, letter

Description automatically generatedIt can be proved a schedule is serializable if and only if the precedence graph has no directed cycle.

Two schedules are view equivalent if they produce the same results.

A non serial schedule is view serializable if it is view equivalent to a serial schedule. Conflict serializable => view serializable but the inverse is not true!!

### Concurrency

Two main approaches to ensure consistency when executing transactions concurrently

* Conservative (pessimistic) methods
  + Actively avoid conflicts
  + Delay (or restart) transactions when they are in conflict
* Optimistic method
  + Assume transactions are rarely in conflict
  + Check for conflicts just before transaction commits

The two main conservative methods are locking and timestamping.

Lock: when a transaction accesses the database, the lock denies access to other transactions, preventing incorrect results. This is widely used to ensure serialisability. A transaction can keep two types of lock:

* Shared (read) lock:
  + The transaction is allowed to read some data item. Any other data item can only read this item.
* Exclusive (write) lock:
  + Transaction is allowed to read/write on some data item. Any other data item has no access to this item.

A transaction requests a lock from the DBMS. The DBMS grants the lock; otherwise the transaction waits. A transaction releases a lock on a data item.

If the item is not currently locked by another transaction, the lock will be granted. If the item is locked, the DBMS checks compatibility between the requested lock and the current existing lock.

A shared lock that has been requested on an existing shared lock will be granted. Otherwise, the transaction must wait. A transaction holds the lock until it explicitly releases it, during execution or when the transaction terminates. To ensure the Isolation property, the effects of a write operation are only made visible when the exclusive lock is released.

Some systems allow a shared lock to be upgraded to an exclusive lock, and an exclusive lock to be downgraded to a shared lock.

Two Phase Lock (2PL) – all locking operations occur before unlocking.

Growing phase: acquire all locks needed, no unlock

Shrinking phase: release all locks, no new lock.

If every transaction in a schedule follows 2PL then the schedule is always conflict serialisable.

Cascading rollback problem: transaction rolls back after a long time, can cause a pile of rollbacks. This is an inefficient database! The solution is rigorous 2PL- release all locks when the transaction commits.

Strict 2PL – release all write locks when the transaction commits, all other locks can be released earlier.

Deadlocks: two or more transactions can wait forever waiting for each other. This happens with the 2PL protocol, also when all locks are released at commit. There are several ways to handle deadlocks

* Timeouts
  + The transaction requests a lock only for a system-defined maximum period of time. After that point, the DBMS assumes a deadlock has occurred, requests a timeout and the transaction rolls back and restarts.
* Deadlock detection

Diagram

Description automatically generated

BFS is used to detect a directed cycle. If you check too often for deadlocks, there is a large computational overhead, leading to a slow database. If you check too rarely, deadlocks can go undetected for a long time.

Parameters for recovery:

* How far to rollback? The simplest thing to do is to roll the entire transaction back. The more efficient thing is to roll only some of the transaction back.
* Choice of deadlock victim
* Avoid starvation: starvation when a specific transaction is always the victim. The solution to this is to count how many times a transaction is aborted. When a limit is reached, use a different selection criteria.

Graphical user interface, text, application

Description automatically generatedAnother locking paramter is granularity- how large is the data item to be locked each time? Coarser large locked items/ fewer locks requested => a lower degree of concurrency is permitted. For finer data item sizes – smaller locked items / more locks requested => more locking information needs to be stored. Granularity affects efficiency – the best item size depends on the nature of transactions.

A recoverable schedule is a schedule that ensures once a transaction T is committed, it should never be necessary to rollback T. Recoverable schedules guarantee durability of transactions. Durability – effects of a committed transactions are permenantly recorded – ACID.

A transaction T reads from transaction T’ in a schedule S if some data item x is first read by T’ and then read by T. A schedule S is recoverable if no transaction T commits in S unless first all transactions T’ commit, from which it reads.

Cascading rollback: a single transaction failure (rollback) leads to a series of transaction rollbacks. A schedule is cascadeless if cascading rollbacks cannot occur. For each pair of transactions Ti and Tj: if Tj reads Ti then the commit operation of Ti appears before commit operation of Tj.

In a recoverable schedule, the commit operation of Ti appears before commit operation of Tj. Cascadeless => recoverable but the inverse is not true.

In summary: cascadeless – read operations are always before commits.

Concurrency control:

Schedules must be serializable and recoverable (DB consistency) and preferably cascadeless (efficiency). Conservative (pessimistic) methods – delay conflicting transactions.

Locking – make conflicting transactions wait.

Timestamping – roll back and restart conflicting transactions.

Timestamping:

No locks means no deadlocks.

Timestamp: unique identifier TS(Ti) that indicates the relative starting time of a transaction Ti. Usually the timestamp is generated by the system clock or by incrementing a logical counter for every new transaction. Timestamp determine the serializability order (order of transactions).

WTS(x): writing time stamp – largest timestamp of a transaction that executed write(x) successfully.

RTS(x): reading time stamp – largest timestamp of a transaction that executed read(x) successfully.

The protocol ensures conflicting read and write operations are executed in timestamp order.

Suppose T issues a read(x):

If TS(T) < WTS(x), read operation rejects, T rolled back. An earlier transaction is trying to read a value of an item that has been updated by a later transaction.

Otherwise, read operation is executed, RTS(x) = max{TS(T), RTS(x)}

Suppose T issues a write(x):

If TS(T) < RTS(x), a later operation that T uses the current value of x and it would be dangerous for T to update it. T is rolled back.

If TS(T) < WTS(x), T attempts to write an obsolete value of x. T is rolled back.

Otherwise, write(x) is executed, WTS(x) = TS(T).

Correctness: timestamping protocol guarantees serializability. There is no cycle in the precedence graph. The timestamping method ensures freedom from deadlock (no transactions waits) and the schedule may not be recoverable, thus not cascadeless (why?).

Problems with timestamping protocol:

Suppose Ti aborts, but Tj has read a data item written by Ti. Tj must abort; if Tj has been allowed to commit the the schedule is not recoverable. Any transaction that reads an item written by Ti must reject. This can lead to cascading rollbacks.

The solution is to perform all writes at the end of processing. All writes of a transaction form an atomic action; no transaction may execute while a transaction is being written. A transaction that aborts is restarted with a new timestamp.

Thomas’ write rule

When a transaction T issues a read(x), same as before

When a transaction T issues a write(x):

If TS(T) < RTS(X): same as before – T rolled back

If TS(T)<WTS(x): ignore the write(x)

Otherwise as before: write(x).

A later transaction already updated the value of x => T attempts to write an outdated value of x which will never be read.

Thomas’ write rule produces a view serializable schedule which can never be produced by previous protocols. This leads to more concurrency and greater efficiency.

Multiversion timestamps are another generalistion of the timestamp protocol. Many items can simultaneously access item x but each of them work on a different version of x. The versions are labelled by timestamps. This increases concurrency. Write(x) either creates a new version of x, or aborts and restarts. Read(x) selects the appropriate version of x. It is always successful. Each version of x contains content (value of xk), WTS(xk) – the timestamp of the transaction that creates version of x, and RTS(xk) – largest timestamp of a transaction that read version xk. When a transaction creates a new version of x, WTS = RTS = TS(T).RTS updates whenever T reads xk and TS(T)>RTS(xk). Let xk be the version of x where WTS(xk) is the largest write timestamp that is <= TS(T).

T issues a WTS(X):

If TS(T) < RTS(x) then T is rolled back: the T that read xk will never see the update of T, there is no serializability.

If TS(T) = RTS(x), x is overwritten because xk was created by the same T

If TS(T) > RTS(x), a new version of x is created.

Versions can be deleted when they are no longer needed. Keep only one version of x older than TS(T), in case T is rolled back.

Checking for view serializability is NP Hard, but conflict serializability is polynomial time.

Optimisitc methods for concurrency control have three phases:

* Read
  + Runs from the beginning of the transaction to just before the commit.
  + All items are read from the DB, and are stored in local variables
  + Only the local copies are updated (not the real data items)
  + Both read and write operations occur, but only in the virtual DB.
* Validation phase
  + After the read phase
  + Checks whether conflict serializability will occur if a commit occurs.
  + If something looks suspicious, abort and restart, discarding the local copies of the data items. For read only transactions, check if the read values are still current; if they aren’t then abort.
  + For transactions that update values, check whether serializability will be maintained. If so, move to the write phase.
* Write phase (write operations only)
  + Only after successful validation
  + Apply local copies to database

The validation phase: Each transaction gets 3 timestamps: at the start of execution, start(T), start of validation phase, validation(T), and at finish time, finish(T). The first check for validation: all transactions S with an earlier start time stamp finish before T starts: finish(S)<start(T). If start(T)<finish(S), check all data written by S were not read by T and S completes the write phase before T enters the validation phase ie start(T)<finish(S)<validation (T). This guarantees T has read current values and write of T are done serially. These are too strong guarantees for serializability- transactions restart more often than needed. Rollbacks involve only a local copy of data, so there are no cascading rollbacks. More accurate validation tests woulf imply more processing time for each transaction and be useless, assuming conflicts are rare. If conflicts are not rare, conservative methods may be faster as conflicts are prevented from happening: locking might be faster than repeating transactions. If optimistic methods are used, more transactions are made to abort if there is a conflict.

Flat transactions have a single start and end point. They are suitable for traditional databases. In modern databases, transactions have a long duration, meaning they are more susceptible to failure, loss of significant amount of work and unacceptably long delays. Ideally the database would be recovered to a stately held shortly before the crash. Further problems with flat transactions include a large number of data items accessed (many items looked for extended periods to preserve transaction isolation, leading to limited concurrency), and deadlocks being likely.

There are several ways to mitigate these

* Nested transaction model
  + Transactions contain other transactions, each with their own start and termination. Only leaf sub-transactions perform database operations. Child transactions start and terminate before their parent. This is known as bottom-up execution. Aborts and commits happen at the top level. Abort undoes all updates within the parent, and commit permanently records updates. The updates of a committed sub-transaction is visible only to its immediate parent. The commit is conditional to the commit/abort of its ancestors. If a subtransaction aborts, the parent node is informed, and can choose how to proceed. Possible actions include: retrying the transaction, ignore the failure (for non vital subtransactions), running a contingency subtransaction, and aborting.
  + Advantages of the nested transaction model include: modularity, finer level of granularity (so easier to handle for concurrency/recovery), subtransactions running concurrently, aborted subtransactions rollback without side effects. This mitigates the effects of a long duration transaction.
  + Can be simulated with save points: identifiable points in flat transactions represent a partially consistent state, and can be used as internal restart points, leading to a finer unit of recovery for the whole transaction.
* Sagas
  + Ordered sequence of flat transactions tht can be interleaved with each other. For every subtransaction Ti, there is a compensentory transaction Ci that undoes the effects of Ti. T1,..Tn if successful, T1,..Tn,Cn,..,C1 if aborts or fails. This relaxes the isolation property, as partial results are revealed to other transactions before they complete. Sagas are useful when sub-transactions are relatively independent and compensating transactions can be produced. Finding compensating transactions can be difficult.
* Dynamic restructuring
  + Split-transaction operation: transaction T is split into two serializable transactions A, B (only possible if serializability is guaranteed). Their actions and resources are divided, and new transactions proceed independently. Partial results of T become visible to other transactions. Conditions for split-transaction operation: if both A and B write to the same object, B’s write operation follow A’s write operation. A cannot see any results of b, but B may see results of A. A is serializabe before B, so if A aborts, B aborts. A and B are serializable in any order.
  + Join Transaction operation: reverse of split-transaction. This merges ongoing work of 2 transactions as though they have been a single transaction.
  + The two operations can be combined: split into A,B and then join B with C. This transfers resources among transactions without making them available to other transactions.
  + Advantages:
    - Adaptive recovery: allows part of the work if a transaction to commit, so not affected by subsequent failures.
    - Reduced Isolation: resources released by committing part of a transaction. This mitigates the effects of a long-duration transaction.
* Workflow models
  + An activity for coordinated execution of multiple tasks by people or processing systems. Workflow models are complex models for any specific busiess applications. They hardly conform to any ACID property.

### Distributed Transactions

A transaction in a DDBMS is initiated in one of the sites and is divided into subtransactions (one at each site). The DDBMS must ensure synchronization of subtransactions with other local transactions, and the ACID property of the transaction. All centralised problems still exist, and a new one appears: multiple copy consistency problem – when an item is updated, it must be updated at every site. Otherwise the global database would be inconsistent.

A global schedule is serializable if each local schedule is serializable (at each site) and the local serializabillity orders (of the transaction) are identical, ie all subtransactions appear at every site in the same order in the equivalent serial schedule.

Text, letter

Description automatically generatedGiven a distributed non-serial schedule, we can test conflict serializability using the precedence graph – schedule is serializable if and only if the precedence graph has no directed cycle. The technical problem is that the database is distributed. No site has full information about all conflicts; building the precedence graph is not trivial. If all transactions are local and have no replication, all centralised (local) concurrecy protocols work eg 2PL and timestamping.

LM – lock manager

Centralised 2PL: treat the database as if it were centralised. Advantages include easy implementation and its easy to detect deadlocks (build wait-for graph of the DDBMS at the central LM). Disadvantages include bottlenecks (overloaded central LM, scalability issues) and low reliability (failure of central LM freezes the whole database).

Primary copy 2PL: many LM across DDBMS, each one responsible for locking a different set of data items. For every replicated data item x, one copy is chosen as the primary copy, and others are slave copies. When x is updated, the local TC locates the primary copy of x then sends a write-lock requiest to the appropriate LM. Only the primay copy needs to be locked and updated. The change propagates to slave copies. This is very efficient when large updates are infrequent (high locality of reference) or when sites do not need the most updates version of data. The load is distributed to many LM so the bottleneck problems are resolved. The reliability problems still remain: there is a large degree of centralisation and the failure of one LM freezes some parts of the DDBMS. All primary copies of this LM are inaccessible. The solution is to have a back up site for each LM (B-LM). When a LM receives an update request, a copy of the request is sent to its B-LM. If the LM doesn’t send a quick update notification, the B-LM assumes the LM failed and acts in place of the LM: sends a copy of the request to its B-LM, notifies everybody that is is the new LM, and performs all updates of the original LM. When the LM recovers, everybody is notified it is again the LM and receives the log of updates made by the B-LM.

Distributed 2PL: there is a LM at every site managing locks for the data at this site. If data is not replicated, same as primary 2PL. Otherwise, the read-once-write-all (ROWA) rule is followed: only a read lock at one site that keeps the item and a write-lock at every item that keeps the item. The copies in these sites must be locked before the update.

To check if a write-lock can be granted, there is a high communication cost. The requesting site waits for confirmation from all sites that keep the item.

|  |  |
| --- | --- |
| Distribute 2PL | Primary copy 2PL |
| High communication costs | Low communication costs |
| Always current values | Potentially outdates values |
|  | When needed, every value can synchronize with the primary copy. |

Majority 2PL is a special case of distributed 2PL. The write-lock is granted if at least half the sites confirm it. The lock holder notifies everyone that it has the lock. Read-lock can be simultaneously held by many users. The write-lock is held by only one user at a time.

### Distributed concurrency control and distributed deadlock schemes

Centralised deadlock detection: a single site is the deadlock detection coordinator (DDC). This maintains the global wait-for graph (WFG). Periodically very site sends its local WFG to the DDC (only differences from the previous local WFG to minimize the communication load) and the DDC updates the global WFG. If a cycle exists, the DDC breaks them and chooses the victims to roll back. If the DDC fails, the whole system is less reliable.

### Hierarchical deadlock detection:

Sites are organised into a binary hierarchy. The sites send their local WFG to the deadlock detection site above them in the hierarchy. L1: leaves, the original sites, perform local deadlock detection. L2 detects deadlocks in pairs of sites of L1. The root detects global deadlocks at all sites. There is no dependency on a centralized detection site, so more reliable and less communication costs but more complex to implement.

Distributed deadlock detection: at every site S, construct the WFG and add on an external node Text. For a transaction Ti in local WFG of S:

* If Ti waits for a lock of a transaction at site S’ add Ti-> Text to S WFG, labelled S’.
* If a transaction waits for a lock of Ti at S’, add Text->Ti to local WFG of S and label edge S’.

At every site if local WFG has a cycle without Text, there is a local deadlock at S. If there is a cycle with Text, there is a potential global deadlock. To find out, merge cycles (using edge-labels to / from Text) until a global cycle is found, or there is no cycle anymore. More robust to failures, but more communication complexity.

Failures in distributed systems (eg message loss, communication link failure, site failure):

Network partitioning: network split into two or more partitions. It is difficult to distinguish whether a communication link or site has failed. Consider a global transaction, acting at many sites. Sub-transactions in one partition decide to commit, and abort in the other, leading to a violation of atomicity of a global transaction. To ensure atomicity, all sub-transactions commit/abort and then the global transaction commits/aborts. In every global transaction, one site acts as a coordinator (transaction manager) (usually where transactoin is initiated), all other sites are participants. The coordinator knows all participants, but a participant may only know the coordinator.

2 Phase commit (2PC): the phases are the voting phase and the decision phase. The coordinator asks participants “Are you ready to commit?”. If one participant votes to abort (or has no timely response) then all participants must abort. If all participants vote to commit, they must all then commit. If a participant votes to abort, it can do so immediately / before voting (unilateral abort). The global transaction then aborts. There is no unilateral commit (could cause inconsistency). If a participant votes to commit, they must wait for a broadcast from the coordinator (global commit or global abort) before taking action – they are blocked. No unilateral abort if someone votes to commit. If a participant doesn’t receive the broadcast / coordintor doesn’t receive some vote, termination protocol is invoked, as a failure is assumed to have happened. The protocol assumes each site keeps a local log, so they can roll back and commit reliably.

Coordinator procedure:

* Voting phase: write *begin\_commit* record to the log file, send *PREPARE* message to all participants and wait for them to respond within the timeout period.
* Decision phase:
  + If a participant replies abort, then write abort record to the log file, send *GLOBAL ABORT* to all participants and wait for acknowledgements.
  + If all participants reply commit, return *READY\_COMMIT* to log file, send *GLOBAL COMMIT* to all participants, and wait for acknowledgements.
  + When all acknowledgements are received, write *end\_transaction* to log file. If a site doesn’t send ack, keep resending global decision until an ack is received.
  + If a site doesn’t vote timely, global abort.

Participant procedure:

* When *PREPARE* is received, write *write\_commit* to log and send *READY\_COMMIT*, or write abort record to log and send *ABORT.*
* If *GLOBAL\_ABORT* is received, write abort to log, abort sub-transactions and send ack
* If *GLOBAL\_COMMIT* is received, write commit to log, commit sub-transactions and send ack.

Diagram

Description automatically generated

2PC termination protocols: invoked when the coordinator or participant fails to receive an expected message, and times out.

Coordinator: timeout can only happen in WAITING or DECIDED. WAITING: cannot commit (not all votes have been received), so global abort. DECIDED: keep sending global decision to sites that don’t ack.

Participants: timout during INITIAL: abort sub-transactions. Timout during PREPARED: participant is blocked.

2PC recovery protocols are invoked when an operational site restarts after failure. Coordinator: failure during INITIAL – recovery just starts commit procedure. Timeout during WAITING – restart commit procedure. Timeout during DECIDED: if all acks are received, commit the transaction. If not, initiate termination protocol.

Participants: failure during INITIAL: unilaterally abort sub-transactions. Timeout during PREPARED- initiate termination protocol. Timeout during PREPARED – initiate termination protocol. Timeout during ABORTED/COMMITTED – no action needed, sub transactions already completed.

## Networks

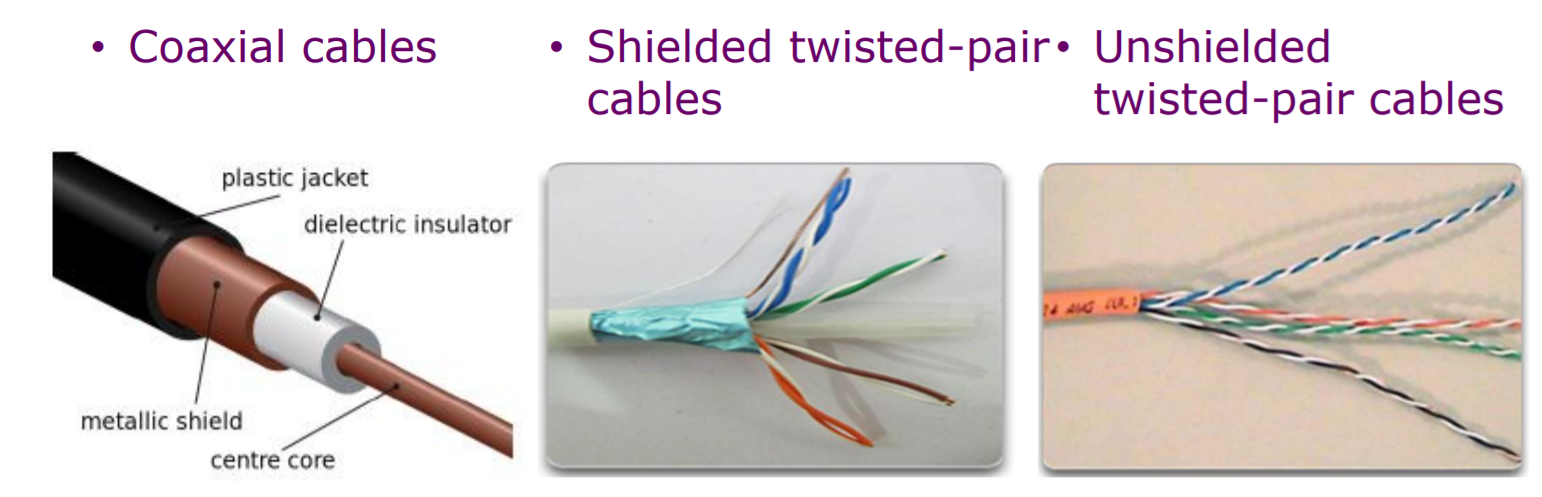
Computer network: group of devices that are connected to one another in order to exchange information or share resources.

Networks are made up of hardware components:

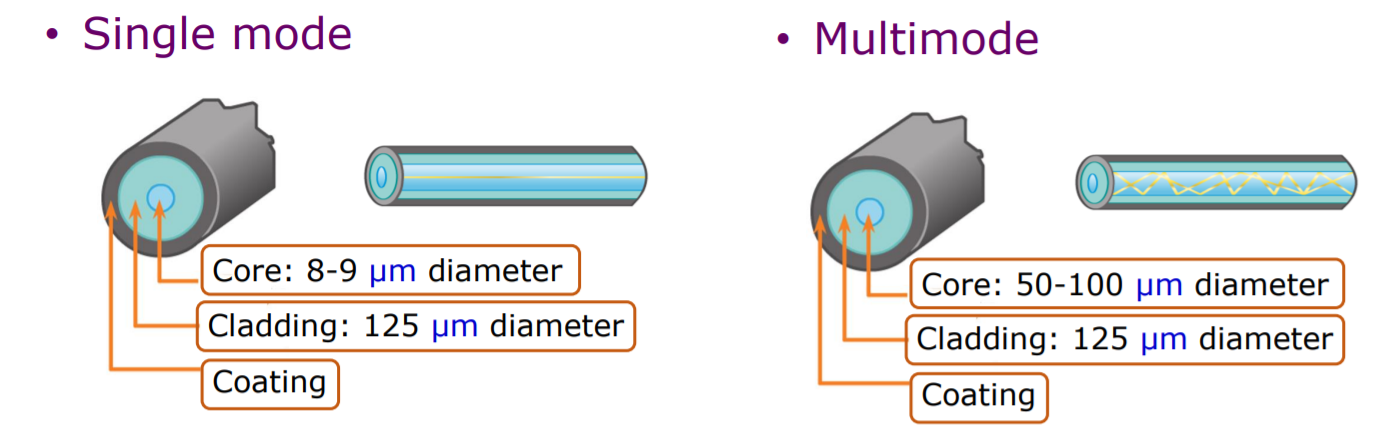
* End devices (end systems/ end stations / hosts) form the interface between users and the underlying communication network. Eg computers, servers, printers
* Intermediary devices interconnect end devices eg routers (connect networks), switches (connect devices within a network), firewalls
* Transmission media provides the channel over which the message travels from source to destination. Data appears as signals in transmission media. Eg copper cables, optic fibres, radio frequency.

Wired Media:

* Copper cables: 3 types

Coaxial cables are cheapest. There is one wire so one bit at a time is sent. Shielded cables are the most expensive. There are 8 wires so 8 bits are sent at once. The shield protects from electric interference from other devices. The unshielded cables are the most common, having an acceptable performance (transfer rate) and are cheaper than the shielded cables. Copper cables transmit electric signal pulses. Transmit rates vary depending on category: cat 3 = 10Mbit/s, cat 5 = 100Mbit/s, cat 6 = 1000Mbit/s.

* Optic fibres

Optic fibres transmit light pulses. They have a high transmit rate (100 Gbit/s). There is a low error rate- repeaters are spaced far apart. If the light refracts more, the light loses energy.

Wireless media:

* Radio frequencies: low frequency, large waves, passes through big obstacles. The signal is carried in electromagnetic spectrum. There are no physical wires. There are 3 groups:
  + Very short distance (Bluetooth, 5-10m)
  + LAN (wifi, 10-few hundred metres)
  + Wide area (cellular / mobile, 10s of miles)
* Satellite radio channels: there are two types:
  + Geostationary: used when cable-based access or other wireless connections are unavailable. Around 36000km above Earth, stationary
  + Low-earth orbiting: closer to earth, moves over the surface up to 2000km above earth.

Protocols define the format and order of messages sent and received among the network entities and actions taken on message transmission and reciept. They allow a network to function properly.

Protocols must account for the following requirements:

* Identified sender/reciever
* Common language and grammar
* Speed and timing of delivery
* Confirmation of requirements

Protocol examples:

* Message timing
  + When someone is able to send a message
  + How much information can be sent and the speed at which it can be delivered
  + How long to wait for responses and action if timeout occurs
* Message delivery options: unicast (1:1), multicast (small group:1), broadcast (everyone)
* Message encoding: converting information into another form for transmission eg converting to radio waves or light signals.
* Message formatting and encapsulation: placing a message within another.
* Message size: breaking up a long message

Access networks

An access network is a type of network which physically connects an end system (computer etc) to the immediate router on a path from the end system to any other distant end system. The following are examples of access networks.

* Digital subscriber line (DSL)
  + Use existing telephone line to central office DSLAM (Digital-Subscriber-Line-Access-Multiplexer)
  + Asysmmetric access – downstream (< 10Mbps) and upstream (< 1Mbps) are different
* Cable network
  + Hybrid fibre coax (HFC) – asymmetric again: downstream (42.8 Mbps) and upstream (30.7 Mbps)
  + Network of cable, fibre attaches home to ISP router. Homes share access network to cable headend – BUS TOPOLOGY
* Ethernet (enterprise access networks)
  + Widely used in companies and universities.
  + Transmission rates 10Mbps, 100 Mbps, 1 Gbps, 10 Gbps
  + Typically end systems connected by ethernet switches
* Home networks, wireless access networks
  + Shared wireless access network connects end system to router via base station (access point).
  + Wireless LANs: within building (around 100 ft) – WIFI
  + Widea area wireless access – provided by mobile operator – 4G/5G. 1-10Mbps

Network Security (consided in all layers)

Packets are analysed (packet sniffing) on broadcast media. All packets are analysed by the network interface. Wireshark is an open source packet sniffer.

Circuit switching:

Two network nodes establish a dedicated communication channel (circuit) before nodes may communicate. Users remain connected and enjoy the full bandwidth of the channel until the session is terminated. Commonly used in traditional telephone networks.

Packet switching:

Transmission links are shared by multiple users. End devices break application data into small pieces for transmission (packets) – this uses transmission links more effectively. Different packets can take different paths to reach the destinations.

Each layer of the open systems interconnection (OSI) model provides functionality to the upper layers and relies on the lower layers. The upper layers are used for user services and applications, and the lower layers are used for hardware and transmission of data.

1. Application layer- emails/web browser/ skype - HTTP
2. Presentation layer – translate data formats / data encryption - DNS
3. Session layer – establish or terminate a session / sync communicating nodes – FTP
4. Transport layer – support communications between applications across networks – TCP/UDP
5. Network layer – determine the best path through the network – Ipv4 / IPv6
6. Data-link layer – define frames / detect and manage collisions / error detection - ethernet
7. Physical layer – manage physical medium and signal transmissions - ethernet

Advantages of a layered model:

* Prevent tech / capability changes in one layer from affecting other layers
* Provide common language to describe networking functions and capabilities
* Assisting in protocol design

Table

Description automatically generated with medium confidenceGraphical user interface, application

Description automatically generatedProtocol de-capsulation (reciever) vs protocol encapsulation (sender side)

Network structure:

The edge of a network is made up of end devices – clients / servers. A client is a traditional / modern user devices. A server is often in data centres. The network core is a mesh of interconnected routers. Packets travel from one router to another, across links on path from source to destination.

Data is broken into small pieces (packets) at network edges. Packet switching enables multiple network edges to share the network core.

Routing: routers determine the next hop on which to send a piece of data.

Forwarding: intermediary network devices move a piece of data from one interface to another.

Routing algorithms select the best paths for packets to reach the destinations. Routers record the next hops and associated interfaces in their routing tables.

Store and forward: telecommunications technique in which intermediary device keeps information and sends it at a later time to the destination or to another intermediary device.

Nodal processing delay: time spent on checking error bits and determining output link; intermediary devices generate the delays. Queuing delays happen at routers.

At a router, if the data input rate is greater than the data output rate, packets will queue up to be transmitted. If the queue fills up, packets are dropped. Lost packets may be retransmitted by previous nodes or the source or not at all.

Transmission delay (store-and-forward delay): amount of time to push all bit of a piece of data onto the wire.

For an L bit packet, the transmission delay is 2L/R where R is in bit/s.

The entire data packet must arrive at the router before it can be transmitted on the next link. Data transmission rate of link = bandwidth.

Propagation delay: time required to travel through medium (in seconds)

Packet delay: dp = dproc + dqueue +dtrans + dprop where dproc is the nodal processing delay, dqueue is the queueing delay, dtrans = L/R = packet length / link bandwidth and dprop = d/s = length of physical link / propagation speed

Traceroute program provides delay measurement from source to router along end-end internet path towatds destination.

For all routers r:

Sends 3 packets to r on path towards destination, r returns packets to sender and sender calculates the time interval between transmission and replies.

### HTTP and the application layer

There are two application architectures: client-server (eg HTTP/ DNS) and Peer to Peer/ P2P.

In a client-server architecture, the client is a a device that requests information or services. They may be intermittently connected and have dynamic IP addresses. Clients do not communicate with each other, only the server. The server is a device that repsonds to requests by providing data / services. A server has a fixed IP address (so clients know where to find it).

In a P2P architecture, there is no dedicated server. Each peer (connected device) can function as both server and client roles are set on a per request basis. There is a P2P index server which holds which device has which information (bit like the yellow pages). A P2P is self scalable – new peers bring new service capability and demands. Peers are intemittently connected and can change IP address.

A process is a program running within a host.

Within the same host, two processes communicate using inter-process comunication defined by the OS. Processes on different hosts communicate by exchanging messages.

A socket is a software mechanism that allows a process to create and send messages into and receive messages from the network.

A socket address is a combination of an IP address and a port number. A process sends and recieves messages to and from its socket.

An application needs certain transport services (reliability, timing, security).

The TCP protocol:

* Connection-oriented: set-up required between client and server processes.
* Reliable transport between sending and receiving processes
* Flow control: sender won’t overwhelm the reciever
* Full duplex connection (can send and receive):connection can send messages to each other at the same time.

UDP protocol

* Unreliable data transfer between sending and receiving processes
* Does not provide : reliability, flow control, congestion control, security, connection setip
* Short-delay transmissions between sending and receiving processes.

There are two types of application protocols: open protocols and proprietary protocols. Open protocols are standard, and anyone can use them. They are defined in request for comments (RFC). They allow for interoperability eg HTTP and SMTP. Examples of proprietary protocols include Skype and Zoom.

HTTP:

HTTP is an application layer protocol for web services. It follows a client (browser)- server (web server) model; anything can be a client or web server. TCP is used at the transport layer. The client creates a socket at port 80, and the server accepts the TCP connection. HTTP messsages are exchanged then the TCP connection closes. HTTP is stateless, the server maintains no information about past client requests.

Round trip time (RTT): time for a small packet to travel from the client to the server and back.

Non persistent HTTP: at most 1 object is sent over a TCP connection, and then the connection closes. Downloading multiple objects requires multiple connections. The response time is 2RTT + file transmission time; one RTT to initiate TCP connection and one for the HTTP request

Persistent HTTP: multiple objects can be sent over one single TCP connection between the client and the server. The response time is RTT + file transmission time, assuming the connection to the server is already established and all files are requested in parallel.

### The transport layer

Two main protocols work at the transport layer: Transmission control protocol (TCP) and User datagram protocol (UDP). Some common services offered by these protocols include providing logical communicaations between application processes running on different hosts, and segmentation:

* Sender side breaks application messages into segments, and passes them to the network layer
* The reciever side reassembles segments into messages, passes them to the application layer.

Applications on both sides use port numbers to set up the communiction. The transport layer uses port numbers and rely on and enhance network layer services. The network layer uses IP addreses for logical communication between hosts. The data-link layer uses MAC addresses to find the destination machine.

16 bits are used for port numbers. Port numbers are split into 3 groups: well-known, registered, private and/or dynamic ports.

|  |  |
| --- | --- |
| TCP | UDP |
| Connection oriented: three way handshake (sender -> destination, ACK-> sender, sender-> destination) before starting data transmission; handshake on both sides to terminate a connection | Connectionless |
| Reliable transport: reorder segments, retransmit loss segments, acknowledgement and timer | Unreliable data transfer – doesn’t use a checksum or similar because it doesn’t care if a datagram was corrupted |
| Flow control: window size to control the maximum number of data bytes that can be sent in one go; avoid overwhelming the reciever | Other services, such as flow control, are unavailable. |
| Full-duplex connection: connection can send messages to each other at the same time | Unreliable datagram |
| Reliable, byte stream oriented |  |

The host uses IP addresses and port numbers to direct segment to the appropriate socket. Each socket has a unique identifier.

Graphical user interface, diagram, table

Description automatically generated with medium confidenceUDP datagram structure and encapsulation

The sender explicitly attaches IP destination address at the network layer and the port number at the transport layer to each packet. The reciever extracts the sender IP address at the network layer, and the port number at the transport layer from the received packet.

Diagram, table

Description automatically generatedTable

Description automatically generatedTCP segment structure and encapsulation

TCP provides a reliable, in-roder byte stream transfer (“pipe”) between the client and the server.

1. The server process must first be running and have created a socket that accepts clients’ connection requests.
2. The client contacts the server by creating the TCP/IP socket, specifying the IP address and port number of the server process. When the client creates a socket, the client TCP establishes a connection to the server TCP.
3. When contacted by the client, the sever TCP creates a new socket for the server process to communicate with that specific client.
   1. Allows server to talk with multiple clients
   2. Source port numbers used to distinguish different processes on the same clients.

TCP and UDP extend IP delivery service between hosts to deliver service between processes.

Multiplexing at sender: handle data from multiple sockets, add transport header

Demultiplexing at receiver: use header information to deliver received datagrams/segments to correct socket.

Connectionless multiplexing/ demultiplexing

When the host sends a UDP segment, all sockets have host-local port number, which is assigned automatically or via bind().

When the host receives a UDP segment, it checks the destination port number in the datagram and directs the UDP datagram to the socket with that port number.

Diagram, timeline

Description automatically generatedCharacteristics of an unreliable channel will determine the complexity of the reliable data transfer (Rdt). A reliable protocol (such as TCP) may be implemented on top of an unreliable end-to-end network layer, such as IP.

Reliable transfer over UDP:

* Add reliability at application layer
* Application- specific error recovery.

Rdt is a stop-and-wait protocol. Only one packet is sent, wait for an ACK and then send the next packet.

Rdt 1.0:

Diagram

Description automatically generated with medium confidenceReliable transfer over a reliable channel. The underlying channel is perfectly reliable – physical cables are used. There are no bit errors and no packets are lost. There are separare FSMs for sender and receiver.

There is one state for both. The sender waits for data from the upper layer. Once received, data is encapsulated with the transport layer header added and the protocol header and the data link layer header and trailer. Data is sent after encapsulation. Once the data is received by the network, the data is extracted from the packet and delivered to the upper layer.

Rdt2.0

There can be bit errors – use checksum (in trailer) to detect bit errors. To deal with errors:

ACKS: receiver explicitly tells the sender the segments received are ok.

NAKS: negative ACKS, explicitly tell the sender segments have errors. Sender retransmits on receipt of a NAK.

Using ACK and NAKs are ARQ (Automatic repeat request):

* Error detection – sender embeds extra bits in data piece
* Feedback – receiver provides feedback (positive and negative)
* Retransmission – sender retransmists erroneous data pieces.

Diagram

Description automatically generated

Diagram

Description automatically generatedDiagram

Description automatically generatedRdt2.0 has some problems. What if the ACK/NAK has been corrupted? Add a checksum to the ACK/NAK. The sender sends one packet, and waits for receiver response. The sender retransmits current packet if ACK/NAK has been corrupted. A sequece number is added to each packet. The receiver discards any duplicate packets. This is rdt2.1 and rdt 2.2.

Wait for feedback

Encapsulate and add checksum if +ve feedback

Diagram

Description automatically generated

Rdt3.0 is used on channels with errors and lost packets, due to congestion. There is a new assumption: the underlying channel can also lose packets. The sender waits a reasonable amount of time for an ACK, and retransmits if no ACK is received in that period. If the packet or ACK is delayed, the receiver must specify the sequence number of the packets being ACKed. This requires a countdown timer.

Pipelined protocols:

A buffer is needed at the receiver to reorder packets. Lots of packets are sent at once.

* Range of sequence number must be increased
  + May be multiple in-transit unacknowledged packets
  + Unique sequence number
* Multiple packets buffering at the sender and/or receiver.
  + Sender buffers packets that have been transmitted but not yet ACKed.
  + Receiver buffers correctly received packets before it can process them to the upper layer.
* Buffering requirements depend on the manner in which a data transfer protocol responds to a lost / corrupted/ overly delayed packet.

There are two generic types of pipelined protocols:

* Go-back-N
  + The sender sends multiple packets without waiting for ACK.
  + Can have up to N unpacked (unacknowledged) packets on the pipeline
  + Receiver sends cumulative ACKs – doesn’t send ACK if there is a gap.
  + Sender has a timer for oldest unpacked packets.
  + More efficient
* Selective repeat
  + Can have up to N unpacked packets in pipeline
  + Individual ACK for each packet
  + Sender maintains timer for each unpacked packet.
  + When timer expires, retransmit only the unpacked packets. If the window size = |range of packet sequence| then duplicate data will be accepted as new.Can be fixed by increasing range of packet sequence numbers.

### Network layer

Routing is the process of discovering network paths.

* Decide what to optimise – eg fairness vs efficiency
* Model the network as a graph of nodes and links
* Update routes for changes in topology

If router J is on the optimal path from I to K, then the optimal path from J to K is on the same route.

* Identify the optimal path from source to destination

Sink tree: optimal routes from source to destination.

Distance metrics include number of hops or time delay.

Dijkstra’s algorithm is used to compute a sink tree on a graph.

Chart, radar chart

Description automatically generatedChart, radar chart

Description automatically generatedDistance vector routing protocols:

Text

Description automatically generatedEach node maintains a table (vector of best distance to destination). Tables are updated by exchanging information between nodes periodically. The table has two entries: the outgoing line (node) and the estimated distance.

A picture containing graphical user interface

Description automatically generatedFailures can cause distance vector to count to infinity, and get caught in a loop while seeking an unreachable node. Good news of a path spreads quickly, whereas bad news of a failure in the network spreads slowly.

Link state routing

A screenshot of a computer

Description automatically generated with medium confidenceList neighbours and weights of links to reach nodes. The challenge is getting packets to all routers. The solution is flooding – a method to send a packet to all network nodes. Each node floods a new packet received on an incoming link by sending it out on all of the other links. Nodes keep track of flooded packets to stop the flood, by using sequence numbers. Flooding does not rely on routing tables being built first. Hierarchical routing reduces the work of route computation but may result in slighty longer paths than flat routing. Only allow flooding in small groups. This reduces network traffic and the cost of computing distances.

Diagram

Description automatically generatedBroadcast routing – broadcast a packet to all nodes.

Diagram, schematic

Description automatically generatedReverse path forwarding (RPF): arrived packets are checked to see if they arrived from a preferred link (link normally used for sending packets towards the source of the broadcast).

### Medium Access Control (MAC) sublayer

The data-link layer is split into logical link control (responsible for error control and flow control) and MAC (responsible for multiple access resolutions ie who transmits next). The key issue is that there is a single physical layer medium for networks communications but multiple devices all want to use it at once. The transmissions interfere with each other and become corrupted.

Static channel allocation:

* Time division multiplication (TDM)
  + Each user gets entire transmission capacity for a fixed time interval
* Frequency Division Multiplexing
  + Each user gets a portion of transmission capacity for whole time.
* The limitations of static channel allocation are:
  + Only works for a few fixed users
  + Data traffic often very bursty
  + If many users do not use their allocated channel capacity – most channels will be idle most of the time.

Dynamic channel allocation

* No user is assigned fixed frequency / fixed time slot
* All users dynamically assigned frequency or time slot, depending upon user requirement.
* Protocols types:
  + Random Access Protocols
  + Controlled Access Protocols
  + Limited contention protocols – combination of first two
  + Channelization protocol – for wireless networks.

Random Access Protocols:

Any station can send data at any time. Station makes decision on whether to send data. Depends on the state of the channel. There is no scheduled time for a station to transmit. Stations can transmit in any order. There is no rule deciding which station should send next. If 2 stations transmit at the same time, there is a collision and the frames are lost.

* Aloha: protocol developed for communication over radio link. A collision occurs when 2 stations transmit simultaneously.
  + Pure Aloha: stations transmit frames whenever they have frames to send. Collisions occur, and frames are lost, when 2 stations transmit simultaneously, or 2 frames try to occupy the channel at the same time, or if the first bit of a frame overlaps with the last bit of another frame.
  + Slotted Aloha: time is divided into frame-size slots. The station can only send a frame at the beginning of a slot, and only one frame can be sent in each slot. If any station is not able to place a frame onto a channel at the beginning of the slot, it has to wait until the next time slot. There is still a possibility of collision if 2 stations try to send frames at the same time.
* CSMA – Carrier Sense Multiple Access, CD – Collision Detection, CA – Collision Avoidance. CSMA tries to minimise collisions. The station checks if the network is busy – senses carrier / channel before transmitting a frame. The chances of collision greatly reduce if a station checks before sending. The propagation delay means there is still a chance of collision still exists.
  + 1-persistant (greedy): send as soon as the network is idle. The channel is continually monitored. If the channel is idle. If the channel is idle, transmit with probability 1. If the channel is busy, wait until it is idle again. When an idle channel is detected, immediately send a frame. When a collision occurs the station waits a random amount of time and then starts again. This protocol has the highest chance of collision.
  + Non-persistant: wait a random time and try again. If the channel is idle, send immediately. If the channel is busy, wait a random amount of time and try again. The random amount of wait time reduces the chance of collisions. It is unlikely two or more stations will wait the same amount of time. This protocol introduces longer delays.
  + P-persistant: used in slotted channels eg slotted aloha. If the channel is busy, wait until the next time slot and start over. If the channel is idle, with probability p transmit and with probability (1-p) defer until the next slot and start over.
* CSMA/CD – sense the channel whilst data is being transmitted. If there is a collision, about the transmission and wait a random time. The transmitting station releases a jam signal- this alerts other stations a collision has occurred. Wait a random time and try again. CSMA/CD improvement is to detect / avoid collisions. Contention times are reduced which improves performance.

Controlled Access Protocol (CAP) – the stations consult each other to find which station has a right to send. They are collision free protocols. A station cannot send unless it has been authorized by other station(s).

* Bitmap: collisions are avoided entirely. Senders must know when it is their turn to send. Before sending any data, all stations state if they have data. Senders 0,1,..,n send their status one-by-one in order. Sender which anounced they had data send in turn. This whole process is repeated.
* A picture containing necklet, accessory

  Description automatically generatedToken passing: tokens sent round the ring defines the sending order. Station with token ay send a frame before passing the token.
* Binary countdown: improves on bitmap protocol. Stations send their addresses in contention slot. The channel ORs bits; stations give up when they send a 0 but receive a 1. A station that see its full address is the next to send.

### Data-link layer

The data-link layer deals with putting data on the transmission medium, framing and error control.

On the sender side, the data link layer accepts packets from the network layer and encapsulates them into frames. These frames are then sent onto the physical layer, which is responsible for transmission of raw sequences of bits. On the receiver side, the data-link layer accepts bits from the physical layer and forms frames, which are sent onto the network layer. The network layer de-encapsulates frames to important packets.

Framing is important so the network layer knows when one frame ends and another begins.

The flag byte is a special byte stream pattern (01111110). This indicates the start and end of a packet.

* Byte count: a frame begins with a count of a number of bytes in in (including the byte saying this, so n-1 bytes of data). This method is simple but hard to resynchronise after an error – bits can be flipped so data bytes and header bytes can be mixed up. Eg if a bit gets flipped in the header byte, the number of data bytes in the frame will be wrong, which then causes the rest of the sequence of frames to be wrong.
* Byte stuffing – useful if the flag value occurs natually in the data. An escape byte is put before the flag sequence, to indicate that it is data and not the end of a frame. If an escape byte occurs natually, an escape byte is put before this too. Occurrences of flags in the data must be *stuffed*. This takes longer than byte count but easy to resynchronise after an error. As data is added, overheads are created, which can build up quickly.
* Bit stuffing – within the data, a 0 is added after each sequence of 5 consecutive 1s – this removes void instances of the flag in the data. Upon receiving 011111: if the next bit is 0 then the stuffed bit is removed. If the next bit is 1: check the next bit. If it is 0 then this is the end of frame marker. If it is a 1 then an error has occurred (can’t have 7 1s in a row).

Error detection and correction: data may be retransmitted if protocol is allowed. EDC: error detection and correction bits. D: data protected by error checking, may include header bytes.

* Text

  Description automatically generated with medium confidenceCodewords are calculated by XORing bits. A frame consists of m data (message) bits and r redundant (check) bits. A n-bit codeword with n =m+r. Find the Hamming distance between two codewords (or count the number of 1s in the record). This gives a simple way to add check bits and correct up to a single bit error.
  + Encoding: number the bits starting from 1 and skip powers of 2 (reserved for parity bits).
  + Decoding: calculate all parities. If all ok, no error. Otherwise add up positions of incorrect ones to work out the position of the error.

Error detection

* Parity: checks errors. XOR – odd parity. Parity bits are appended to the data. Single parity bit detects single bit errors. 2D bit parity detects and corrects single bit errors – can be done using checksum. Even parity: total number of 1s in the string (including the parity bit) is even. Odd parity: total number of 1s in the string (including the parity bit) is odd.
* Checksum: the sender divides the data into k segments of m bits and sums the segments. The 1s complement of the sum forms the checksum. This is sent to alongside the other segments. The receiver sums all the received segments, including the checksum segment, and computs the 1s complement. If the result is 0 then there are no errors.
* Text

  Description automatically generatedCyclic Redundancy Check (CRC): stored in the trailer of the frame, the crc uses manipulations with polynomials. This is one of the most commonly used error detection codes. Let M1 (the message) be the message of n bits. M1 is padded with CRC code and sent. The CRC is generated using a given polynomial (1011). P1 is agreed between the sender and receiver. The receiver receives M1+CRC and extracts M1 using P1: divide the received message by the divisor. If the result is 0, the message is correct. Otherwise an error has occurred.

### Wireless MAC

Still works on the data-link layer.

Wireless hosts run applications and may be stationary or mobile. A base station is typically connected to wired networks. They are responsible for sending packets between wired networks and wireless hosts in its “area”. Similar to intermediary devices in wired networks. Wireless links are used to connect mobiles to base stations. They are also used as backbone links. Multiple Access protocol coordinates link access.

Wireless network modes:

* Infrastructure mode: base stations connect wireless hosts to wired networks. Handoff: mobile hosts change base stations providing connections to wired networks.
* Ad hoc mode has no base stations. Nodes can only transmit to other nodes within link coverage. Nodes organise themselves into a network route among themselves.

Differences from wired links (wireless link characteristics)

* Decreasing signal strength: radio signal attentuates as it propagates through matter.
* Interference from other sources: standaried wireless network frequencies shared by other devices. Devices can intefere as well.
* Multipath propagation: radio signal reflects on objects around, arriving at the destination at slightly different times.

Only omnidirectional antenna emit signals in all directions, in a sphere shape.

Graphical user interface, text, application, chat or text message

Description automatically generatedA signal is a time-varying quantity that conveys information.

In a wired connection, high and low voltages give a distinguishable wave form. There is noise in the signal which needs to be removed. Amplitude is used to differentiate between high and low voltage.

Radio waves are a form of electromagnetic radiation with wave forms between 30 and 300GHz. They are generated by oscillating electric fields. In the receiver, the radio signal creates a tiny voltage which can be measured.

Exposed terminals are senders who can sense each other but still transmit safely to different receivers. 802.11 is a standard.

802.11: multiple access. Collisions happen. 802.11 uses CSMA to avoid collisions, but does not detect collisions. It is difficult to receive (sense collisions) when transmitting due to weak received signals (fading). Not all collisions can be detected due to hidden terminals/ fading.

802.11: CSMA/CA

If the sender senses the channel is idle for DIFS (Distributed Inter Frame space) (a time period), the transmit the entire frame. If the channel is busy:

1. Start a random backoff timer
2. The timer counts down even when the channel is idle
3. Sense again timer expires
4. If no ACK is received or still busy, increase the random backoff interval and repeat the process.

CSMA/CA cannot avoid hidden terminals. Hidden terminals are senders that cannot sense each other but nonetheless collide at the intended receiver (hidden terminal problem).

CSMA/CA with RTS/CTS

Allow the sender to “reserve” the channel rather than random access of data frames: avoid collisions of long data frames. The sender first transmits small request to send (RTS) packets to a base station or access point using CSMA. Base station broadcasts clear to send (CTS) in response to RTS. The CTS is heard by all nodes. The sender transmits data frame and other stations defer transmissions.

802.11 frame format

Table

Description automatically generated

WLANs Architecture

Service Set Identifier (SSID): logical name identifies a particular wireless network. A base station/ access point broadcasts beacon frames periodically. Such advertisements can include the SSID associated with this base station / access point. A client device scans all available frequencies in search of SSIDs to join. A client selects the SSID to join a wireless network.

Basic Service Set (BSS) aka “cell” in infrastructure contains a base station or access point with one or more client devices. This is the basic building block of an 802.11 network. Client devices communitcate through the base station or access point. Client devices connected to a BSS are associated. The association is dynamic: client devices can move in or out of the coverage, or be switched off.

Diagram

Description automatically generated with low confidence802.11 Association

The access point (AP) admin chooses the frequency for the access point. Inteference is possible: channel can be same as that chosen by the neighbouring AP. The host must associate with an AP.

The host:

1. Scan channels, listening for beacon frames containing AP’s name (SSID) and MAC address (BSSID).
2. Select AP to associate with ie select WLAN to join.
3. May perform authentication

Passive / active scanning

Diagram

Description automatically generated

### Physical layer

There are two distinct senses of bandwidth: bit rate (transmission capacity, rate of data transmission) and width of range of frequencies, as used by a signal.

Baseband: range running from 0 to some max frequency, typically applicable to wired media.

Passband: signals occupying some range of frequencies, as would pass through corresponding frequency filters.

Available band width: range of frequencies usefully transmissible in a medium. A physical property of the transmission medium.

Digital signals are obtained from an analog signal: information transmitted by varying some physical property, such as voltage or current. Digital signals are encoded by (eg) low and high. Encoding schemes:

* Non return to zero: NRZ
  + +ve=1, -ve = 0. Voltage does not return to 0, changes only when the bit value changes. Relies on sender and reciever having accurate, in sync clocks. Transitions can be used to correct small deviations. A problem is long runs of consecutive bit with the same value; constant signal values cannot synchronize the communicating devices.
* Non return to zero inverted: NRZI
  + Tries to aleviate problem in NRZ. 0 = no change in level. 1 = change in voltage – high if current state is low and vice versa. Fixes problem of sending consecutive 1s but not 0s.
* Bipolar encoding, aka alternate mark inversion (AMI)
  + 0 = 0v, 1 = high/low v. Chosen voltage inverted from last transmission of 1
* Manchester encoding
  + Merge explicit clock signal with data signal – XOR to merge. Low->high = 1, high->low = 0. Uses signal changes to transmit data and achieve synchronization. Guaranteed transitions occur with each bit transmitted. But 2x the bandwidth of NRZ is required.

Multiplexing: channels are often shared by multiple signals.

* Frequency divison multiplexing (FDM) – total bandwidth divided into a series of non overlapping frequency bands.
* Wavelength division multiplexing (WDM) – used in fibre optic networks. Different users use different (colour) wavelengths (of light). Each user essentially gets the range of frequencies associated with a colour.
* Time division multiplexing (TDM)- round robin
* Code division multiplexing (CDM) – method allowing every transmitter to use entire channel all the time. Individual transmissions are extracted by a receiver using coding theory. The channel itself merges the transmissions. Each transmitter (station) has a chip (code) – a 4 bit vector made up of 1 and -1. The chips are chosen so they are all othorgonal to each other. Transmit data by transmitting either 1. Their chip sequence (transmit 1), 2. Negation of chip sequence (transmit -1), 3. Nothing if they don’t want to transmit.