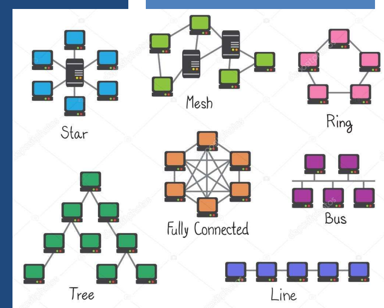
TCSS 558: APPLIED DISTRIBUTED COMPUTING

Distributed Systems: Types and Architectures



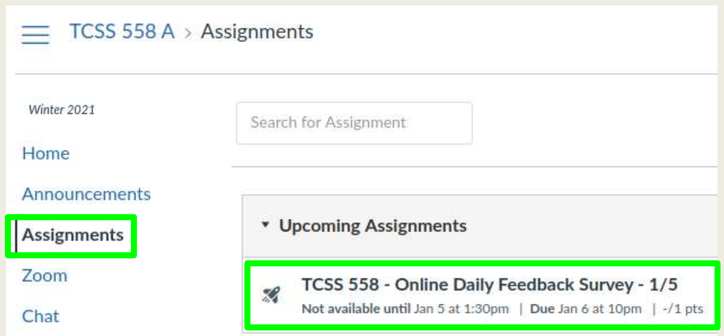
Wes J. Lloyd
School of Engineering & Technology (SET)
University of Washington - Tacoma

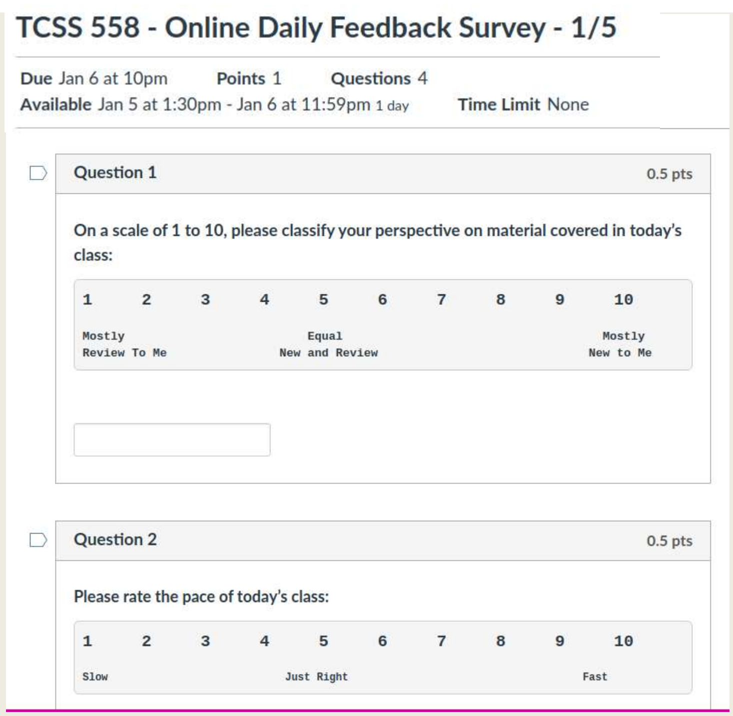
OBJECTIVES – 1/14

- Questions from 1/12
- Assignment 0: Cloud Computing Infrastructure Tutorial
- Chapter 1.3 – Types of distributed systems
  - Pervasive Systems: Sensor networks
- Chapter 2: Distributed System Architectures:
  - Chapter 2.1 – Architectural Styles
  - Layered
  - Object-based
    - Service oriented architecture (SOA)
  - Resource-centered architectures
    - Representational state transfer (REST)
  - Event-based
    - Publish and subscribe (Rich Site Summary RSS feeds)

ONLINE DAILY FEEDBACK SURVEY

- Daily Feedback Quiz in Canvas – Available After Each Class
- Extra credit available for completing surveys ***ON TIME***
- Tuesday surveys: due by Wed @ 10p
- Thursday surveys: due Mon @ 10p





MATERIAL / PACE

- Please classify your perspective on material covered in today's class (23 respondents):
- 1-mostly review, 5-equal new/review, 10-mostly new
- Average – 7.46 (↑ - *previous 6.92*)

- Please rate the pace of today's class:
- 1-slow, 5-just right, 10-fast
- Average – 5.67 (↑ - *previous 5.46*)

FEEDBACK FROM 1/12

- ***My questions is whether message-oriented middleware only supports text based messages, but RPC could send any binary data?***

- This is dependent on the API provided by the messaging system (e.g. Rabbit MQ, Apache Kafka, Amazon SQS)
- Many will support sending files which is binary data

FEEDBACK - 2

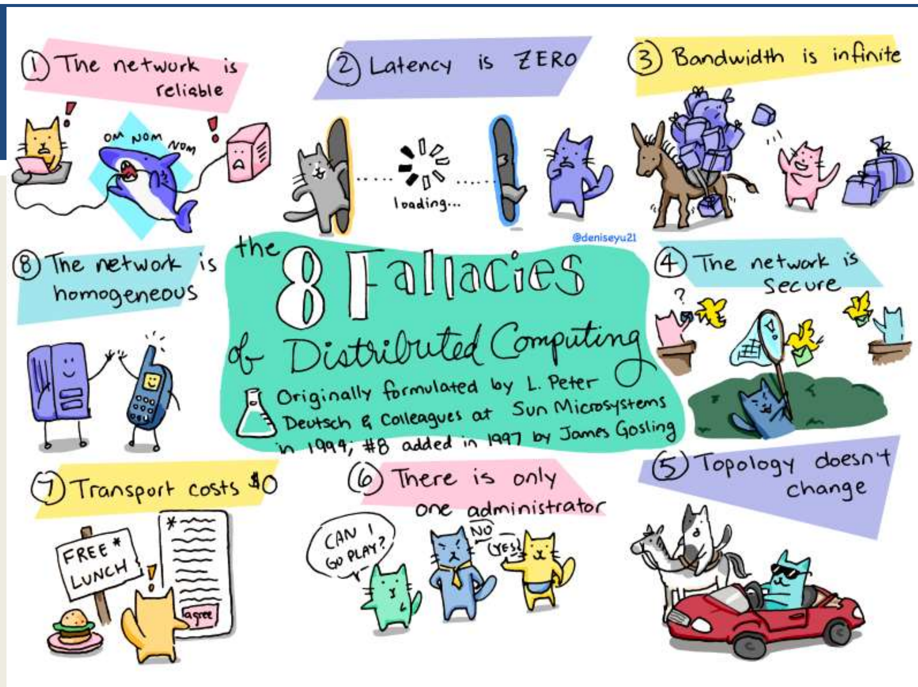
- ***I see the term "middleware" being used repeatedly in Chapter 1 of the book. Can you please help me understand what It means in the context of distributed computing?***
- Middleware typically refers to a software layer or service that provides an interface to enable clients to interact with the components (nodes) of a distributed system
- Middleware exposes an application programming interface (API) that a client application will leverage
- The middleware implement inter-node communication, replication, synchronization, locking and be involved in providing distribution transparency

January 14, 2021	TCSS558: Applied Distributed Computing [Winter 2021] School of Engineering and Technology, University of Washington - Tacoma	L4.7

FEEDBACK - 3

- ***With sensor networks including a large number of IoT devices, are cities today being considered as smart cities or is the idea of smart cities still a while away in the future?***
  - ***I consider things such as everyday objects in street life like sensors on streetlamps, cameras, temperature sensors, and other devices that collect data out in the public.***
- A variety of projects have been implemented around the world
- Adoption levels vary widely in the US, Europe, Asia, etc.
- This article provides a review:
- **Mehmood, Yasir, Farhan Ahmad, Ibrar Yaqoob, Asma Adnane, Muhammad Imran, and Sghaier Guizani. "Internet-of-things-based smart cities: Recent advances and challenges." IEEE Communications Magazine 55, no. 9 (2017): 16-24.**

January 14, 2021	TCSS558: Applied Distributed Computing [Winter 2021] School of Engineering and Technology, University of Washington - Tacoma	L4.8



OBJECTIVES – 1/14

- Questions from 1/12
- Assignment 0: Cloud Computing Infrastructure Tutorial
- Chapter 1.3 – Types of distributed systems
  - Pervasive Systems: Sensor networks
- Chapter 2: Distributed System Architectures:
  - Chapter 2.1 – Architectural Styles
  - Layered
  - Object-based
    - Service oriented architecture (SOA)
  - Resource-centered architectures
    - Representational state transfer (REST)
  - Event-based
    - Publish and subscribe (Rich Site Summary RSS feeds)

ASSIGNMENT 0

- ***Preparing for Assignment 0:***
  - Establish AWS Account
    - Standard account – ***** request cloud credits from instructor *****
      - Specify "AWS CREDIT REQUEST" as subject of email
      - Include email address of AWS account
    - **AWS Educate Starter account** – some account limitations
      - https://awseducate-starter-account-services.s3.amazonaws.com/AWS_Educate_Starter_Account_Services_Supported.pdf
  - Establish local Linux/Ubuntu environment
- Task 1 – AWS account setup
- Task 2 – Working w/ Docker, creating Dockerfile for Apache Tomcat
- Task 3 – Creating a Dockerfile for haproxy
- Task 4 – Working with Docker-Machine
- Task 5 – For Submission: Testing Alternate Server Configurations

OBJECTIVES – 1/14

- Questions from 1/12
- Assignment 0: Cloud Computing Infrastructure Tutorial
- Chapter 1.3 – Types of distributed systems
  - **Pervasive Systems:** Sensor networks
- Chapter 2: Distributed System Architectures:
  - Chapter 2.1 – Architectural Styles
  - Layered
  - Object-based
    - Service oriented architecture (SOA)
  - Resource-centered architectures
    - Representational state transfer (REST)
  - Event-based
    - Publish and subscribe (Rich Site Summary RSS feeds)

PERVASIVE SYSTEM TYPE:
SENSOR NETWORKS

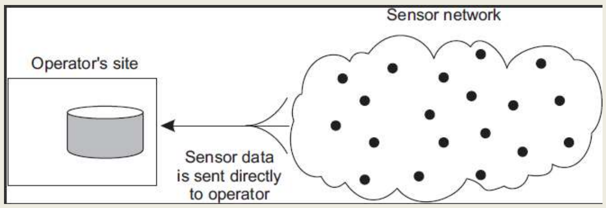
- Tens, to hundreds, to thousands of small nodes
- Simple: small memory/compute/communication capacity
- Wireless, battery powered (or battery-less)
- Limited: restricted communication, constrained power
- Equipped with sensing devices
- Some can act as actuators (control systems)
  - Example: enable sprinklers upon fire detection
- Sensor nodes organized in neighborhoods
- Scope of communication:
  - Node – neighborhood – system-wide

PERVASIVE SYSTEM TYPE:
SENSOR NETWORKS - 2

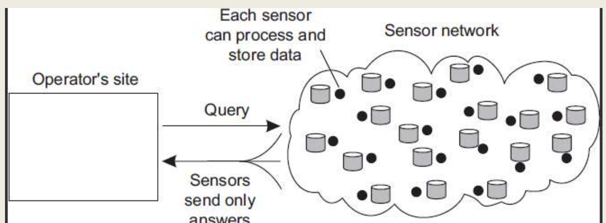
- Collaborate to process sensor data in app-specific manner
- Provide mix of data collection and processing
- <u>Nodes may implement a distributed database</u>
- Database organization: centralized to decentralized
- In network processing: forward query to all sensor nodes along a tree to aggregate results and propagate to root
- Is aggregation simply data collection?
- Are all nodes homogeneous?
- Are all network links homogeneous?
- How do we setup a tree when nodes have heterogeneous power and network connection quality?

CENTRALIZED VS. DECENTRALIZED DATA STORAGE

- Centralized:



- Decentralized:



WHO AGGREGATES AND STORES DATA?

- Consider the **tradeoff space** for:
  - sensor network data storage and processing

Centralized:
- Single point-of-failure
- No node coordination
- No node processing or storage
- "Dumb" nodes
- Less expensive node
- Central server can experience intense network traffic

Decentralized:
- Nodes require high compute power
- "Smart" nodes
- Expensive nodes
- network traffic is distributed

SENSOR NETWORKS - 3

- What are some unique requirements for sensor networks middleware?
  - Sensor networks may consist of different types of nodes with different functions
  - Nodes may often be in suspended state to save power
    - Duty cycles (1 to 30%), strict energy budgets
  - Synchronize communication with duty cycles
  - How do we manage membership when devices are offline?

TYPES OF DISTRIBUTED SYSTEMS

- HPC, Cluster, Grid, Cloud
- Distributed information systems
  - Transactions
  - Application Integration: Shared files, DBs, RPC, RMI, Message-oriented middleware
- Pervasive Systems
  - Ubiquitous computing systems
  - Mobile systems
  - Sensor networks

Identify the type of distributed system: E-commerce website (e.g. eBay, Amazon)

HPC, Cluster, Grid, Cloud

Distributed information system

Pervasive System: ubiquitous computing system

Pervasive: mobile system

Pervasive: sensor network

Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

Identify the type of distributed system: Assisted living home monitoring system for elderly

HPC, Cluster, Grid, Cloud

Distributed information system

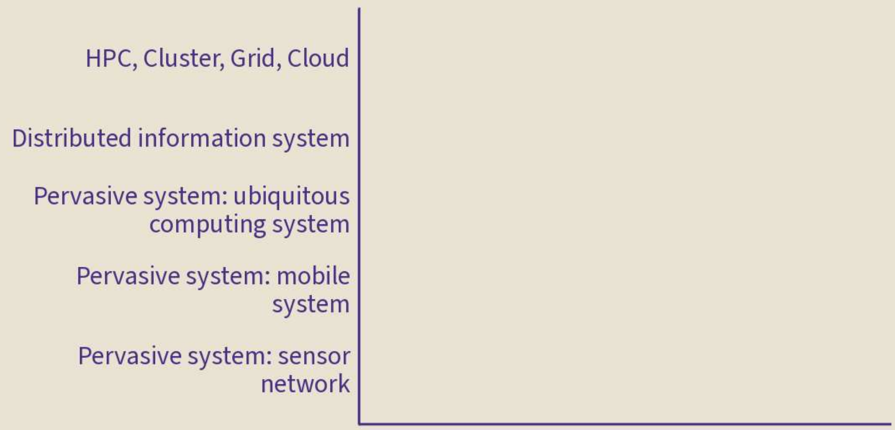
Pervasive system: ubiquitous computing system

Pervasive system: mobile system

Pervasive system: sensor network

Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

Identify the type of distributed system: Seismic monitoring network - warning system for earthquakes



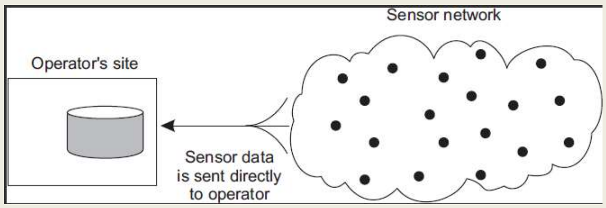
Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

EXAMPLES OF DISTRIBUTED SYSTEMS

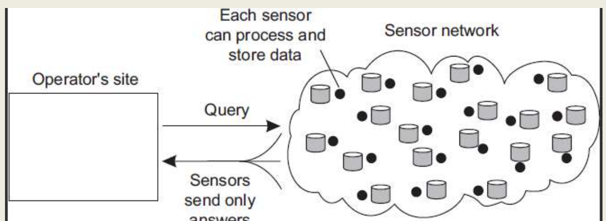
- Classify the following types of distributed systems:
- Web search engine
- Assisted living home monitoring system for elderly
- Ecommerce websites: e.g. eBay, Amazon
- Wikipedia: online encyclopedia
- Amazon Elastic Compute Cloud (EC2)
- Massively multiplayer online games (MMOG)
- Seismic monitoring network: warning system for earthquakes
- Worldwide Large Hadron Collider (LHC) Computing Grid
- Hospital health informatics and records system
- Canvas: web based learning environment
- Modern automobile with self-driving features

WHAT ARE SOME TRADEOFFS FOR CENTRALIZED VS. DECENTRALIZED DATA STORAGE?
EXAMPLE: SENSOR NETWORKS

- Centralized:



- Decentralized:



OBJECTIVES – 1/14

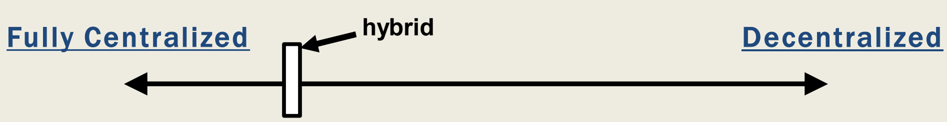
- Questions from 1/12
- Assignment 0: Cloud Computing Infrastructure Tutorial
- Chapter 1.3 – Types of distributed systems
  - Pervasive Systems: Sensor networks
- Chapter 2: Distributed System Architectures:
  - Chapter 2.1 – Architectural Styles
  - Layered
  - Object-based
    - Service oriented architecture (SOA)
  - Resource-centered architectures
    - Representational state transfer (REST)
  - Event-based
    - Publish and subscribe (Rich Site Summary RSS feeds)

DISTRIBUTED SYSTEM ARCHITECTURES

- Provides logical organization of a distributed system into software **components**
- **Logical**: How system is perceived, modeled
  - *The OO/component abstractions*
  - *The "idealists" view of the system*
- **Physical** – how it really exists
  - The "realist" view of the system
- Middleware
  - Helps separate application from platforms
  - Helps organize and assemble distributed components
  - Helps components communicate
  - Enables system to be extended
  - Supports replication within the distributed system
  - Provides "realization" of the architecture

CENTRALIZED VS. DECENTRALIZED DISTRIBUTED SYSTEM ARCHITECTURE

- Tradeoff space: degree of distribution of the system



• Single point-of-failure	• Multiple failure points
• No nodes: vertical scaling	• Nodes: horizontal scaling
• Always consistent	• Eventually consistent
• Less available (fewer 9s)	• More available (more 9s)
• Immediate updates	• Rolling updates
• No data partitions	• Data partitioned or replicated

ARCHITECTURAL BUILDING BLOCKS

- **COMPONENT**: modular unit with well-defined, required, and provided **interfaces** that is replaceable within its environment
- Components can be replaced while system is running
- Interfaces must remain the same
- Preserving interfaces enables interoperability

- **CONNECTOR**: enables flow of ***control*** and ***data*** between components

- Distributed system architectures are conceived using components and connectors

January 14, 2021	TCSS558: Applied Distributed Computing [Winter 2021] School of Engineering and Technology, University of Washington - Tacoma	L4.27

WE WILL RETURN AT 2:42PM

OBJECTIVES – 1/14

- Questions from 1/12
- Assignment 0: Cloud Computing Infrastructure Tutorial
- Chapter 1.3 – Types of distributed systems
  - Pervasive Systems: Sensor networks
- Chapter 2: Distributed System Architectures:
  - Chapter 2.1 – Architectural Styles
  - Layered
  - Object-based
    - Service oriented architecture (SOA)
  - Resource-centered architectures
    - Representational state transfer (REST)
  - Event-based
    - Publish and subscribe (Rich Site Summary RSS feeds)

ARCHITECTURAL STYLES

- Layered
- Object-based
  - Service oriented architecture (SOA)
- Resource-centered architectures
  - Representational state transfer (REST)
- Event-based
  - Publish and subscribe (Rich Site Summary RSS feeds)

OBJECTIVES – 1/14

- Questions from 1/12
- Assignment 0: Cloud Computing Infrastructure Tutorial
- Chapter 1.3 – Types of distributed systems
  - Pervasive Systems: Sensor networks
- Chapter 2: Distributed System Architectures:
  - Chapter 2.1 – Architectural Styles
  - **Layered**
  - Object-based
    - Service oriented architecture (SOA)
  - Resource-centered architectures
    - Representational state transfer (REST)
  - Event-based
    - Publish and subscribe (Rich Site Summary RSS feeds)

January 14, 2021	TCSS558: Applied Distributed Computing [Winter 2021] School of Engineering and Technology, University of Washington - Tacoma	L4.31

DISTRIBUTED SYSTEM GOALS TO CONSIDER

- Consider how architectural style may impact:
- Availability
- Accessibility
- Responsiveness
- Scalability
- Openness
- Distribution transparency
- Supporting resource sharing
- Other factors…

January 14, 2021	TCSS558: Applied Distributed Computing [Winter 2021] School of Engineering and Technology, University of Washington - Tacoma	L4.32

LAYERED ARCHITECTURES

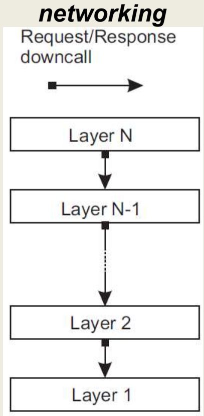
- Components organized in layers
- Component at layer L_j downcalls to lower-level components at layer L_i (where i < j)
- Calls go down
- Exceptional cases may produce upcalls

January 14, 2021	TCSS558: Applied Distributed Computing [Winter 2021] School of Engineering and Technology, University of Washington - Tacoma	L4.33

LAYERED ARCHITECTURES - 2

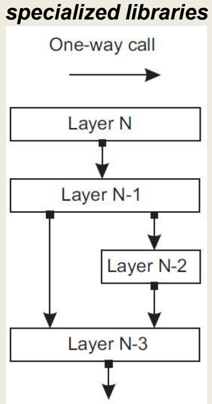
Pure-layered Organization

networking



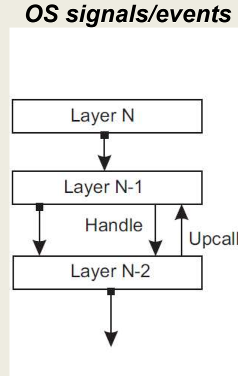
Mixed-layered organization

specialized libraries



Layered w/ upcalls organization

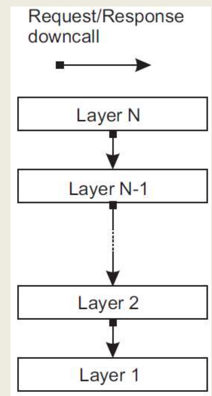
OS signals/events



January 14, 2021	TCSS558: Applied Distributed Computing [Winter 2021] School of Engineering and Technology, University of Washington - Tacoma	L4.34

LAYERED ARCHITECTURES - 3

- Consider an architecture with 5 layers
- Does a client interacting with "Layer 5" of the distributed system need to be concerned with details regarding the implementation of lower layers *(layers 1, 2, 3, 4)* ?

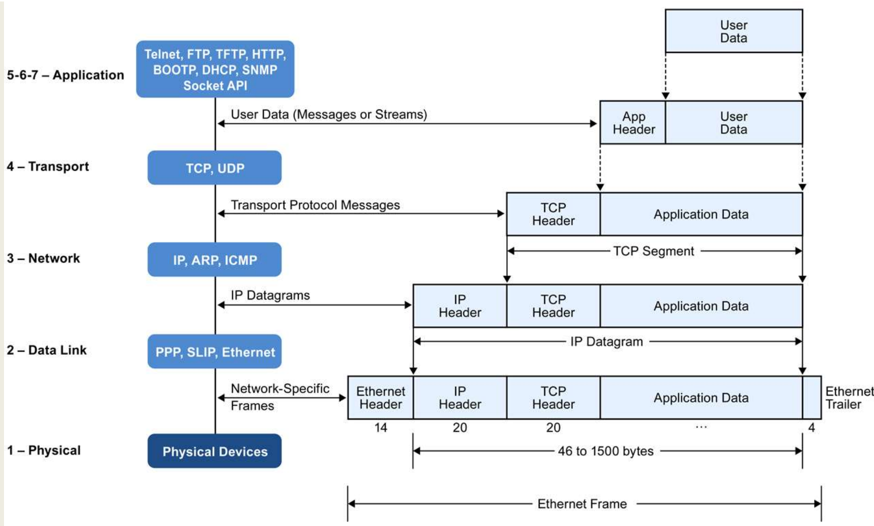


COMMUNICATION-PROTOCOL STACKS

- Example: pure-layered organization
- Each layer offers an interface specifying functions of the layer
- Communication protocol: rules used for nodes to communicate
- Layer provides a **service**
- **Interface** makes service available
- **Protocol** implements communication for a layer

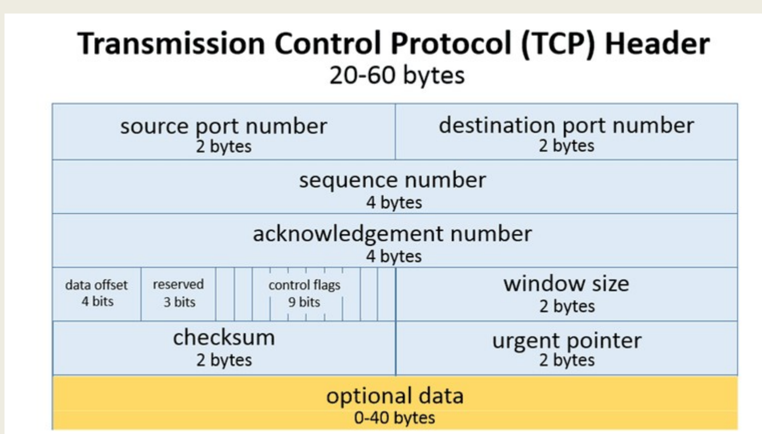
- **New services can be built atop of existing layers to reuse lower level implementation(s)**
- Abstractions make it easier to reuse existing layers which already implement communication basics

HOW A NETWORK PACKET IS BUILT



January 14, 2021	TCSS558: Applied Distributed Computing [Winter 2021] School of Engineering and Technology, University of Washington - Tacoma	L4.37

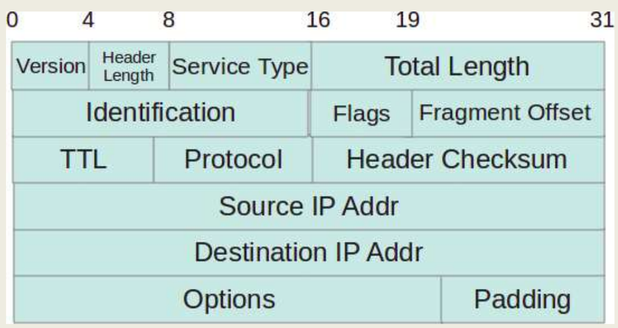
TCP HEADER



January 14, 2021	TCSS558: Applied Distributed Computing [Winter 2021] School of Engineering and Technology, University of Washington - Tacoma	L4.38

IP HEADER

- Source / Destination IP Addr
- IPv4: 32bits / 4 bytes
- IPv6: 128bits / 16 bytes



TRANSMISSION CONTROL PROTOCOL (TCP)

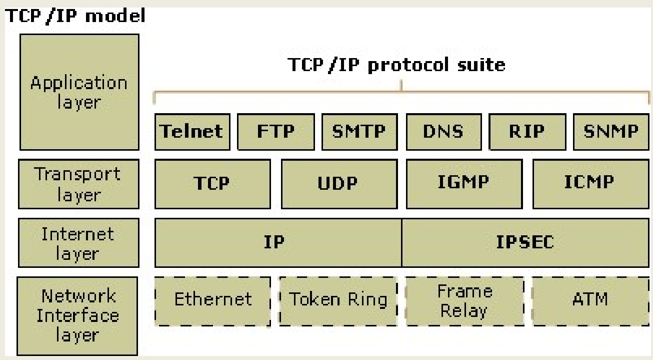
- TCP (layer 4) provides easy to use API
- API supports:
  - setup, tear down of connection(s)
  - sending and receiving of messages
- TCP preserves ordering of transferred data
- TCP detects and corrects lost data

- But TCP is "protocol" agnostic
  - A protocol is a language of messages exchanged to enable communication
  - Application layer communication is programming language agnostic
  - Code can be written in many programming languages to "speak" the "language" of a custom protocol known as an ***APPLICATION PROTOCOL***

- What should the application protocol say?

COMMON APPLICATION LAYER PROTOCOLS

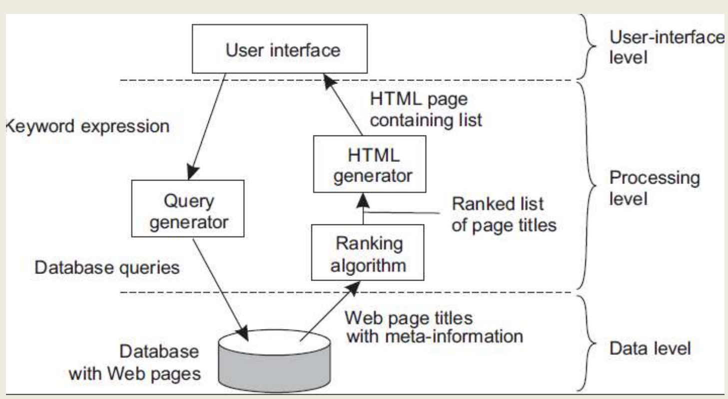
- Telnet, FTP, TFTP, HTTP, DHCP, DNS, NTP, POP, RTP, SMTP, Telnet, RPC, LDAP



January 14, 2021 | TCSS558: Applied Distributed Computing [Winter 2021] School of Engineering and Technology, University of Washington - Tacoma | L4.41

APPLICATION LAYERING

- Distributed application example: Internet search engine



January 14, 2021 | TCSS558: Applied Distributed Computing [Winter 2021] School of Engineering and Technology, University of Washington - Tacoma | L4.42

APPLICATION LAYERING

- Three logical layers of distributed applications
  - The data level
  - Application interface level
  - The processing level

APPLICATION LAYERING

- Three logical layers of distributed applications
  - The data level (M)
  - Application interface level (V)
  - The processing level (C)
- Model view controller architecture – distributed systems
  - Model – database - handles data persistence
  - View – user interface - also includes APIs
  - Controller – middleware / business logic

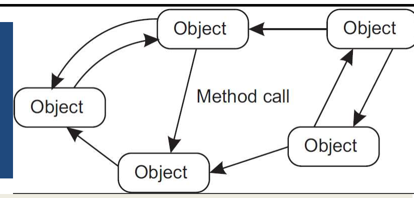
OBJECTIVES – 1/14

- Questions from 1/12
- Assignment 0: Cloud Computing Infrastructure Tutorial
- Chapter 1.3 – Types of distributed systems
  - Pervasive Systems: Sensor networks
- Chapter 2: Distributed System Architectures:
  - Chapter 2.1 – Architectural Styles
  - Layered
  - **Object-based**
    - **Service oriented architecture (SOA)**
  - Resource-centered architectures
    - Representational state transfer (REST)
  - Event-based
    - Publish and subscribe (Rich Site Summary RSS feeds)

OBJECT-BASED ARCHITECTURES

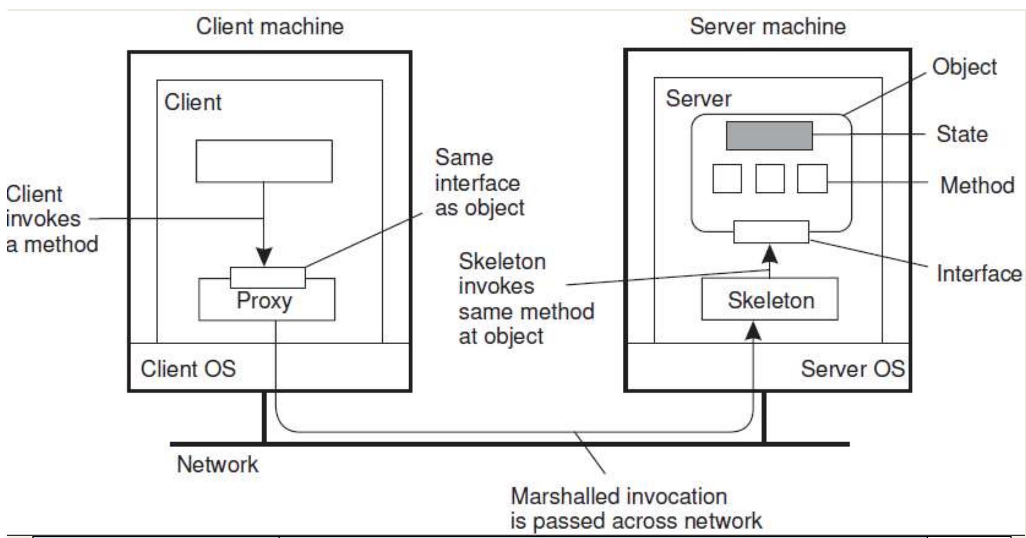
- Enables loose and flexible component organization
- Objects == components
- Enable distributed node interaction via function calls over the network
- Began with C - Remote Procedure Calls (RPC)
  - Straightforward: package up function inputs, send over network, transfer results back
  - Language independent
  - In contrast to web services, RPC calls originally were more intimate in nature
  - Procedures more “coupled”, not as independent
  - The goal was not to decouple and widgetize everything

OBJECT-BASED ARCHITECTURES - 2



- Distributed objects Java- Remote Method Invocation (RMI)
  - Adds object orientation concepts to remote function calls
  - Clients bind to proxy objects
  - Proxy provide an object interface which transfers method invocation over the network to the remote host
- How do we replicate objects?
  - Object marshalling – serialize data, stream it over network
  - Unmarshalling- create an object from the stream
  - Unmarshall local object copies on the remote host
  - JSON, XML are some possible data formats

DISTRIBUTED OBJECTS



DISTRIBUTED OBJECTS - 2

- A counterintuitive feature is that state is not distributed
- Each "remote object" maintains its own state
- Remote objects may not be replicated
- Objects may be "mobile" and move around from node to node
  - Common for data objects
- For distributed (remote) objects consider
  - Pass by value
  - Pass by reference .... *(does this make sense?)*

SERVICE ORIENTED ARCHITECTURE

- Services provide always-on encapsulated functions over the internet/web
- Leverage redundant cloud computing infrastructure
- Services may:
  - Aggregate multiple languages, libraries, operating systems
  - Include (wrap) legacy code
- Many software components may be involved in the implementation
  - Application server(s), relational database(s), key-value stores, in memory-cache, queue/messaging services

SERVICE ORIENTED ARCHITECTURE - 2

- Are more easily developed independently and shared vs. systems with distributed object architectures
- Less coupling
- An error while invoking a distributed object may crash the system
- An error calling a service (e.g. mismatching the interface) generally does not result in a system crash

January 14, 2021	TCSS558: Applied Distributed Computing [Winter 2021] School of Engineering and Technology, University of Washington - Tacoma	L4.51

OBJECTIVES – 1/14

- Questions from 1/12
- Assignment 0: Cloud Computing Infrastructure Tutorial
- Chapter 1.3 – Types of distributed systems
  - Pervasive Systems: Sensor networks
- Chapter 2: Distributed System Architectures:
  - Chapter 2.1 – Architectural Styles
  - Layered
  - Object-based
    - Service oriented architecture (SOA)
  - **Resource-centered architectures**
    - **Representational state transfer (REST)**
  - Event-based
    - Publish and subscribe (Rich Site Summary RSS feeds)

January 14, 2021	TCSS558: Applied Distributed Computing [Winter 2021] School of Engineering and Technology, University of Washington - Tacoma	L4.52

RESOURCE BASED ARCHITECTURES

- Motivation:
  - Increasing number of services available online
  - Each with specific protocol(s), methods of interfacing
  - Connecting services w/ different TCP/IP protocols → integration nightmare
    - Need for specialized client for each service that speaks the application protocol "language"…
- Need standardization of interfaces
  - Make services/components more pluggable
  - Easier to adopt and integrate
  - Common architecture

REST SERVICES

- Representational State Transfer (REST)
- Built on HTTP
- Four key characteristics:
  1. Resources identified through single naming scheme
  2. Services offer the same interface
     - Four operations: GET PUT POST DELETE
  3. Messages to/from a service are fully described
  4. After execution server forgets about client
     - Stateless execution

HYPERTEXT TRANSPORT PROTOCOL (HTTP)

- An ASCII-based request/reply protocol for transferring information on the web
- HTTP request includes:
  - request method (GET, POST, etc.)
  - Uniform Resource Identifier (URI)
  - HTTP protocol version understood by the client
  - headers—extra info regarding transfer request
- HTTP response from server
  - Protocol version & status code →
  - Response headers
  - Response body

HTTP status codes:
2xx — *all is well*
3xx — *resource moved*
4xx — *access problem*
5xx — *server error*

January 14, 2021 | TCSS558: Applied Distributed Computing [Winter 2021] School of Engineering and Technology, University of Washington - Tacoma | L4.55

REST-FUL OPERATIONS

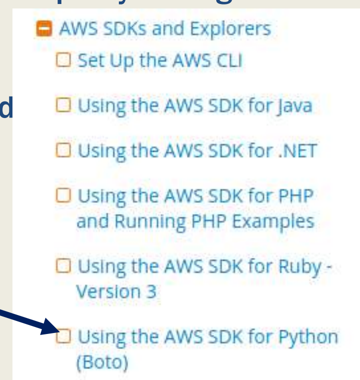
Operation	Description	
PUT	Create a new resource	(C)reate
GET	Retrieve state of a resource in some format	(R)ead
POST	Modify a resource by transferring a new state	(U)pdate
DELETE	Delete a resource	(D)elete

- Resources often implemented as objects in OO languages
- REST is weak for tracking state
- Generic REST interfaces enable ubiquitous *"so many"* clients

January 14, 2021 | TCSS558: Applied Distributed Computing [Winter 2021] School of Engineering and Technology, University of Washington - Tacoma | L4.56

EXAMPLE: AMAZON S3

- Amazon S3 offers a REST-based interface
- Requires signing HTTP authorization header or passing authentication parameters in the URL query string
- REST: GET/PUT/POST/DELETE
- SOAP: 16 operations, moving toward deprecation
- Python boto ~50 operations (SDK for Python)
- SDKs for other languages



January 14, 2021	TCSS558: Applied Distributed Computing [Winter 2021] School of Engineering and Technology, University of Washington - Tacoma	L4.57

REST - 2

- Defacto web services protocol
- Requests made to a URI – uniform resource identifier
- Supersedes SOAP – Simple Object Access Protocol
- Access and manipulate web resources with a predefined set of stateless operations (known as web services)
- Responses most often in JSON, also HTML, ASCII text, XML, no real limits as long as text-based
- curl – generic command-line REST client: https://curl.haxx.se/

January 14, 2021	TCSS558: Applied Distributed Computing [Winter 2021] School of Engineering and Technology, University of Washington - Tacoma	L4.58

```
// WSDL Service Definition
<?xml version="1.0" encoding="UTF-8"?>
<definitions  name ="DayOfWeek"
  targetNamespace="http://www.roguewave.com/soapworx/examples/DayOfWeek.wsdl"
  xmlns:tns="http://www.roguewave.com/soapworx/examples/DayOfWeek.wsdl"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://schemas.xmlsoap.org/wsdl/">
  <message name="DayOfWeekInput">
    <part name="date" type="xsd:date"/>
  </message>
  <message name="DayOfWeekResponse">
    <part name="dayOfWeek" type="xsd:string"/>
  </message>
  <portType name="DayOfWeekPortType">
    <operation name="GetDayOfWeek">
      <input message="tns:DayOfWeekInput"/>
      <output message="tns:DayOfWeekResponse"/>
    </operation>
  </portType>
  <binding name="DayOfWeekBinding" type="tns:DayOfWeekPortType">
    <soap:binding style="document"
      transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="GetDayOfWeek">
      <soap:operation soapAction="getdayofweek"/>
      <input>
        <soap:body use="encoded"
          namespace="http://www.roguewave.com/soapworx/examples"
          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      </input>
      <output>
        <soap:body use="encoded"
        namespace="http://www.roguewave.com/soapworx/examples"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      </output>
    </operation>
  </binding>
  <service name="DayOfWeekService" >
    <documentation>
      Returns the day-of-week name for a given date
    </documentation>
    <port name="DayOfWeekPort" binding="tns:DayOfWeekBinding">
      <soap:address location="http://localhost:8090/dayofweek/DayOfWeek"/>
    </port>
   </service>
</definitions>
```

L4.59

```
// REST/JSON
// Request climate data for Washington

{
 "parameter": [
  {
    "name": "latitude",
    "value":47.2529
  },
  {
    "name": "longitude",
    "value":-122.4443
  }
  ]
}
```

L4.60

OBJECTIVES – 1/14

- Questions from 1/12
- Assignment 0: Cloud Computing Infrastructure Tutorial
- Chapter 1.3 – Types of distributed systems
  - Pervasive Systems: Sensor networks
- Chapter 2: Distributed System Architectures:
  - Chapter 2.1 – Architectural Styles
  - Layered
  - Object-based
    - Service oriented architecture (SOA)
  - Resource-centered architectures
    - Representational state transfer (REST)
  - Event-based
    - Publish and subscribe (Rich Site Summary RSS feeds)

January 14, 2021 | TCSS558: Applied Distributed Computing [Winter 2021] School of Engineering and Technology, University of Washington - Tacoma | L4.61

PUBLISH-SUBSCRIBE ARCHITECTURES

- Enables separation between processing and coordination
- Types of coordination:

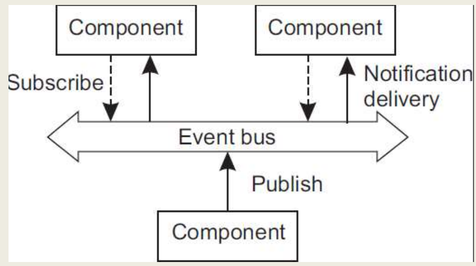
	Temporally coupled (at the same time)	Temporally decoupled (at different times)
Referentially coupled (*dependent on name*)	**Direct** Explicit synchronous service call	**Mailbox** Asynchronous by name (address)
Referentially decoupled (*name not required*)	**Event-based** Event notices published to shared bus, w/o addressing	**Shared data space** Processes write tuples to a shared data space

Publish and subscribe architectures

January 14, 2021 | TCSS558: Applied Distributed Computing [Winter 2021] School of Engineering and Technology, University of Washington - Tacoma | L5.62

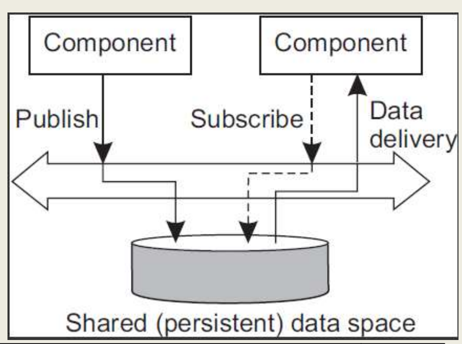
PUBLISH-SUBSCRIBE ARCHITECTURES - 2

- **Event-based coordination**
- Processes do not know about each other explicitly
- Processes:
  - **Publish:** a notification describing an event
  - **Subscribe:** to receive notification of specific kinds of events
- Assumes subscriber is presently up (*temporally coupled*)
- Subscribers must actively **MONITOR** event bus



PUBLISH SUBSCRIBE ARCHITECTURES - 3

- **Shared data space**
- Full decoupling (name and time)
- Processes publish "tuples" to shared dataspace (publish)
- Processes provide search pattern to find tuples (subscribe)
- When tuples are added, subscribers are notified of matches
- **Key characteristic:** Processes have no explicit reference to each other



PUBLISH SUBSCRIBE ARCHITECTURES - 4

- Subscriber describes events interested in
- Complex descriptions are intensive to evaluate and fulfil
- Middleware will:
- Publish matching notification and data to subscribers
  - Common if middleware lacks storage
- Publish only matching notification
  - Common if middleware provides storage facility
  - Client must explicitly fetch data on their own
- Publish and subscribe systems are generally scalable
- What would reduce the scalability of a publish-and-subscribe system?

January 14, 2021	TCSS558: Applied Distributed Computing [Winter 2021] School of Engineering and Technology, University of Washington - Tacoma	L5.65

QUESTIONS

January 14, 2021 TCSS558: Applied Distributed Computing [Winter 2021]
School of Engineering and Technology, University of Washington - Tacoma L4.66