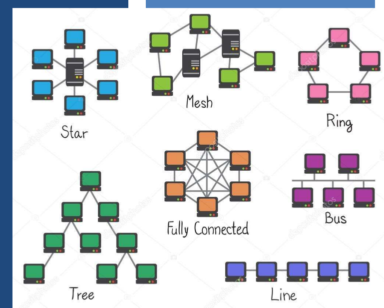
TCSS 558:
APPLIED DISTRIBUTED COMPUTING



Chapter 4 – Communication–III

Chapter 6 - Coordination

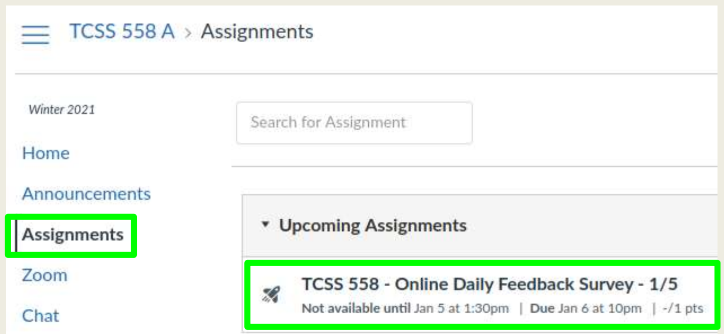
Wes J. Lloyd
School of Engineering
& Technology (SET)
University of Washington - Tacoma

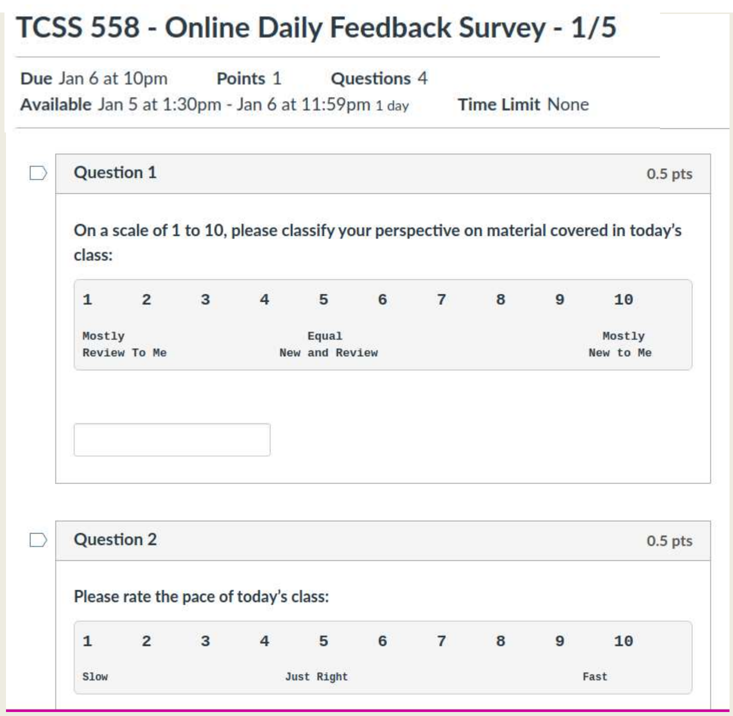
OBJECTIVES – 3/2

- Questions from 2/25
- Assignment 2: Replicated Key Value Store
- Chapter 4: Communication
  - Chapter 4.3: Message Oriented Communication
  - Chapter 4.4: Multicast Communication
- Chapter 6: Coordination
  - Chapter 6.1: Clock Synchronization

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 (15 respondents):
- 1-mostly review, 5-equal new/review, 10-mostly new
- **Average – 6.80 (↑ - *previous 6.11*)**

- Please rate the pace of today's class:
- 1-slow, 5-just right, 10-fast
- **Average – 5.80 (↑ - *previous 5.58*)**

March 2, 2021	TCSS558: Applied Distributed Computing [Winter 2021] School of Engineering and Technology, University of Washington - Tacoma	L15.5

FEEDBACK FROM 2/25

- ***In assignment 2, when a client sends the "exit" command to a node, should only the node who gets the command will be shut down, or should all the nodes be shut down?***

- To implement a distributed exit, the node receiving the exit command would need to relay the "exit" command to every known node.
- A distributed exit command is not described in the assignment.
- **Implementation is optional**

March 2, 2021	TCSS558: Applied Distributed Computing [Winter 2021] School of Engineering and Technology, University of Washington - Tacoma	L15.6

OBJECTIVES – 3/2

- Questions from 2/25
- Assignment 2: Replicated Key Value Store
- Chapter 4: Communication
  - Chapter 4.3: Message Oriented Communication
  - Chapter 4.4: Multicast Communication
- Chapter 6: Coordination
  - Chapter 6.1: Clock Synchronization

March 2, 2021	TCSS558: Applied Distributed Computing [Winter 2021] School of Engineering and Technology, University of Washington - Tacoma	L15.7

SHORT-HAND-CODES FOR MEMBERSHIP TRACKING APPROACHES

- Include readme.txt or doc file with instructions in submission
- Must document membership tracking method

>> please indicate which types to test <<

ID	Description
F	Static file membership tracking – file is not reread
FD	Static file membership tracking DYNAMIC - file is periodically reread to refresh membership list
T	TCP membership tracking – servers are configured to refer to central membership server
U	UDP membership tracking - automatically discovers nodes with no configuration

March 2, 2021	TCSS558: Applied Distributed Computing [Winter 2021] School of Engineering and Technology, University of Washington - Tacoma	L15.8

ASSIGNMENT 2

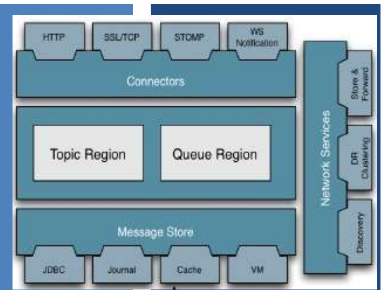
- Sunday March 14th
- Goal: Replicated Key Value Store
- Team signup to be posted on Canvas under 'People'
- Build off of Assignment 1 GenericNode
- Focus on TCP client/server w/ replication
- How to track membership for data replication?
  - Can implement multiple types of membership tracking for extra credit

March 2, 2021	TCSS558: Applied Distributed Computing [Winter 2021] School of Engineering and Technology, University of Washington - Tacoma	L15.9

OBJECTIVES – 3/2

- Questions from 2/25
- Assignment 2: Replicated Key Value Store
- Chapter 4: Communication
  - Chapter 4.3: Message Oriented Communication
  - Chapter 4.4: Multicast Communication
- Chapter 6: Coordination
  - Chapter 6.1: Clock Synchronization

March 2, 2021	TCSS558: Applied Distributed Computing [Winter 2021] School of Engineering and Technology, University of Washington - Tacoma	L15.10



Apache ActiveMQ

CH. 4.3: MESSAGE-ORIENTED COMMUNICATION

L15.11

CHAPTER 4

- 4.1 Foundations
  - Protocols
  - Types of communication
- 4.2 Remote procedure call
- 4.3 Message-oriented communication
  - Socket communication
  - Messaging libraries
  - Message-Passing Interface (MPI)
  - Message-queueing systems
  - Examples
- 4.4 Multicast communication
  - Flooding-based multicasting
  - Gossip-based data dissemination

These sections feature many details, Our focus is on the "big picture"

AMQP PROTOCOL

- Message-queueing systems initially developed to enable legacy applications to interoperate
- Decouple inter-application communication to "open" messaging-middleware
- Many are proprietary solutions, ***so not very open***
- e.g. Microsoft Message Queueing service, Windows NT 1997
- **Advanced message queueing protocol (AMQP)**, 2006
- Address openness/interoperability of proprietary solutions
- Open wire protocol for messaging with powerful routing capabilities
- Help *abstract* messaging and application interoperability by means of a generic open protocol
- Suffer from incompatibility among protocol versions
- pre-1.0, 1.0+

AMQP - 2

- Consists of: Applications, Queue managers, Queues
- **Connections:** set up to a queue manager, TCP, with potentially many channels, stable, reused by many channels, long-lived
- **Channels:** support short-lived one-way communication
- **Sessions:** bi-directional communication across two channels
- **Link:** provide fine-grained flow-control of message transfer/status between applications and queue manager

AMQP MESSAGING

- AMQP nodes: producer, consumer, queue
- Producer/consumer: represent regular applications
- Queues: store/forward messages

- Persistent messaging:
- **Messages** can be marked ***durable***
- These messages can only be delivered by nodes able to recover in case of failure
- Non-failure resistant nodes must reject durable messages
- **Source/target** nodes can be marked ***durable***
- Track what is durable (node state, node+msgs)

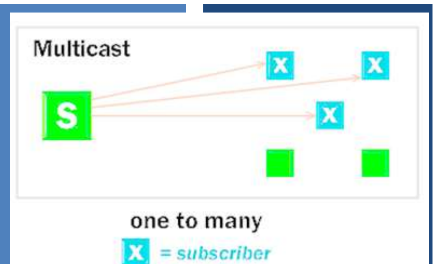
MESSAGE-ORIENTED-MIDDLEWARE EXAMPLES:

- **Some examples:**
- RabbitMQ, Apache QPid
  - Implement Advanced Message Queueing Protocol (AMQP)

- Apache Kafka
  - **Dumb broker** (message store), similar to a distributed log file
  - **Smart consumers** – intelligence pushed off to the clients
  - Stores stream of records in categories called topics
  - Supports voluminous data, many consumers, with minimal O/H
  - Kafka **does not track** which messages were read by each consumer
  - Messages are removed after timeout
  - Clients must track their own consumption (*Kafka doesn't help*)
  - Messages have key, value, timestamp
  - Supports high volume pub/sub messaging and streams

OBJECTIVES – 3/2

- Questions from 2/25
- Assignment 2: Replicated Key Value Store
- Chapter 4: Communication
  - Chapter 4.3: Message Oriented Communication
  - Chapter 4.4: Multicast Communication
- Chapter 6: Coordination
  - Chapter 6.1: Clock Synchronization



Apache ActiveMQ

CH. 4.4: MULTICAST COMMUNICATION

CHAPTER 4

- 4.1 Foundations
  - Protocols
  - Types of communication
- 4.2 Remote procedure call
- 4.3 Message-oriented communication
  - Socket communication
  - Messaging libraries
  - Message-Passing Interface (MPI)
  - Message-queueing systems
  - Examples
- 4.4 Multicast communication
  - Flooding-based multicasting
  - Gossip-based data dissemination

**These sections feature many details,
Our focus is on the "big picture"**

March 2, 2021 | TCSS558: Applied Distributed Computing [Winter 2021]
School of Engineering and Technology, University of Washington - Tacoma | L15.19

MULTICAST COMMUNICATION

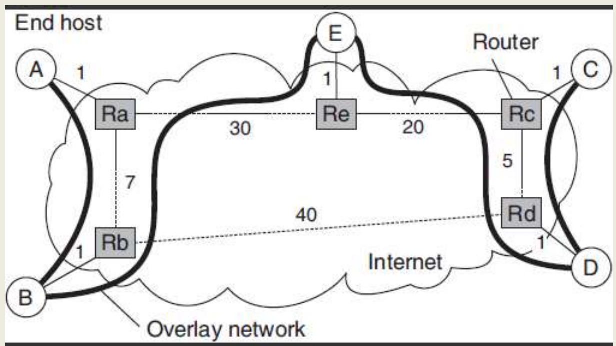
- Sending data to multiple receivers
- Many ***failed*** proposals for network-level / transport-level protocols to support multicast communication
- **Problem:** How to set up communication paths for information dissemination?
- **Solutions:** require huge management effort, human intervention

- Focus shifted more recently to **peer-to-peer** networks
  - Structured overlay networks can be setup easily and provide efficient communication paths
  - Application-level multicasting techniques more successful
  - Gossip-based dissemination: unstructured p2p networks

March 2, 2021 | TCSS558: Applied Distributed Computing [Winter 2021]
School of Engineering and Technology, University of Washington - Tacoma | L15.20

NETWORK STRUCTURE

- Overlay network
  - Virtual network implemented on top of an actual physical network
- Underlying network
  - The actual physical network that implements the overlay



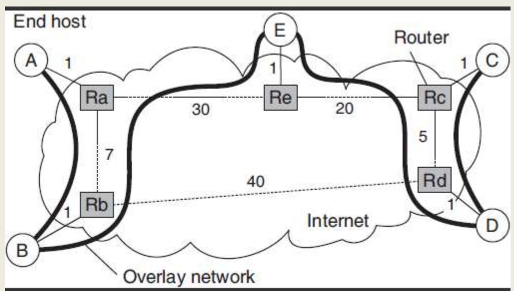
APPLICATION LEVEL TREE-BASED MULTICASTING

- Application level multi-casting
  - Nodes organize into an overlay network
  - Network routers not involved in group membership
  - Group membership is managed at the application level (A2)
- Downside:
  - Application-level routing likely less efficient than network-level
  - Necessary tradeoff until having better multicasting protocols at lower layers
- Overlay topologies
  - TREE: top-down, unique paths between nodes
  - MESH: nodes have multiple neighbors; multiple paths between nodes

MULTICAST TREE METRICS

- Measure quality of application-level multicast tree
- **Link stress:** is defined per link, counts how often a packet crosses same link (***ideally not more than 1***)
- **Stretch**: ratio in delay between two nodes in the **overlay** vs. the **underlying** networks

Numbers represent network delay between nodes



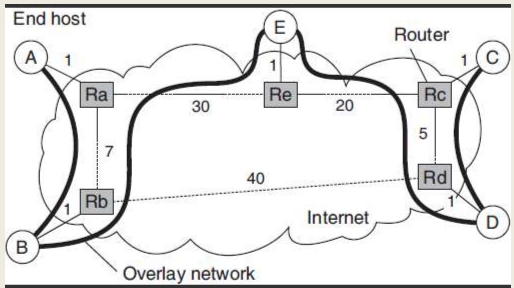
MULTICAST TREE METRICS - 2

- **Stretch (Relative Delay Penalty RDP)**
- CONSIDER routing from B to C
- ***What is the Stretch?***
- Stretch (delay ratio) = Overlay-delay / Underlying-delay
- ***Overlay:*** B→Rb→Ra→Re→E→Re→Rc→Rd→D→Rd→Rc→ C = 73
- ***Underlying:*** B→Rb→Rd→Rc→C = 47
- Stretch = 73 / 47 = 1.55
- Captures additional time (stretch) to transfer msg on overlay net

- **Tree cost:** Overall cost of the overlay network
- Ideally would like to minimize network costs
- Find a minimal spanning tree which minimizes total time for disseminating information to all nodes

FLOOD-BASED MULTICASTING

- Broadcasting: every node in overlay network receives message



- How many nodes are in the overlay network?
- How many nodes are in the underlying network?

FLOOD-BASED MULTICASTING

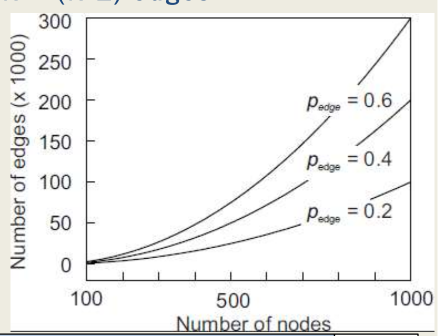
- Broadcasting: every node in overlay network receives message
- Key design issue: minimize the use of intermediate nodes for which the message is not intended
- If only leaf nodes are to receive the multicast message, many intermediate nodes are involved in ***storing*** and ***forwarding*** the message *not meant for them*
- Solution: construct an overlay network for each multicast group
  - Sending a message to the group, becomes the same as broadcasting to the multicast group (*group of nodes that listen and receive traffic for a shared IP address*)
- **Flooding**: each node simply forwards a message to each of its neighbors, except to the message originator

RANDOM GRAPHS

- When there is no information on the structure of the overlay network
- Assume network can be represented as a **Random graph**
- Random graphs are described by a probability distribution
- Probability P_{edge} that two nodes are joined
- Overlay network will have: ½ * P_{edge} * N * (N-1) edges

Random graphs allow us to assume some structure (# of nodes, # of edges) regarding the network by scaling the P_{edge} probability

Assumptions may help then to reason or rationalize about the network…



PROBABILISTIC FLOODING

- *….Washington state in winter?*
- When a node is flooding a message, concept is to enforce a probability that the message is spread (p_{flood})
- Throttle message flooding based on a probability
- Implementation needs to considers # of neighbors to achieve various p_{flood} scores
- With lower p_{flood} messages may not reach all nodes
- **USEFULNESS:** For random network with 10,000 nodes
- With p_{edge} = 0.1 and p_{flood} =.01
- Achieves 50-fold reduction in messages vs. full flooding

PROBABILISTIC FLOODING

- *….Washington state in winter?*
- When a node is flooding a message, concept is to enforce a prob
- Thrott
- Impler … to achiev
- With lower p_{flood} messages may not reach all nodes
- **USEFULNESS:** For random network with 10,000 nodes
- With p_{edge} = 0.1 and p_{flood} =.01
- Achieves 50-fold reduction in messages vs. full flooding

How many edges does network with 10,000 nodes have with p_{edge}=0.1?

March 2, 2021	TCSS558: Applied Distributed Computing [Winter 2021] School of Engineering and Technology, University of Washington - Tacoma	L15.29

PROBABILISTIC FLOODING

- *….Washington state in winter?*
- When a node is flooding a message, concept is to enforce a prob
- Thrott
- Impler … to achiev
- With l… s
- **USEFU**… des
- With p_{edge} = 0.1 and p_{flood} =.01
- Achieves 50-fold reduction in messages vs. full flooding

How many edges does network with 10,000 nodes have with p_{edge}=0.1?

Edges = $\frac{1}{2} * P_{edge} * N * (N-1)$

March 2, 2021	TCSS558: Applied Distributed Computing [Winter 2021] School of Engineering and Technology, University of Washington - Tacoma	L15.30

PROBABILISTIC FLOODING

- *….Washington state in winter?*
- When a node is flooding a message, concept is to enforce a pro
- Throt
- Imple … to achie
- With
- USEF … es
- With p_{edge} = 0.1 and p_{flood} =.01
- Achieves 50-fold reduction in messages vs. full flooding

How many edges does network with 10,000 nodes have with p_{edge}=0.1?

Edges = $\frac{1}{2} * P_{edge} * N * (N-1)$
$\frac{1}{2} * (.1) * (10000) * (9999)$

March 2, 2021	TCSS558: Applied Distributed Computing [Winter 2021] School of Engineering and Technology, University of Washington - Tacoma	L15.31

PROBABILISTIC FLOODING

- *….Washington state in winter?*
- When a node is flooding a message, concept is to enforce a pro
- Throt
- Imple … to achie
- With
- USEF … es
- With p_{edge} = 0.1 and p_{flood} =.01
- Achieves 50-fold reduction in messages vs. full flooding

How many edges does network with 10,000 nodes have with p_{edge}=0.1?

Edges = $\frac{1}{2} * P_{edge} * N * (N-1)$
$\frac{1}{2} * (.1) * (10000) * (9999)$
4,999,500 edges

March 2, 2021	TCSS558: Applied Distributed Computing [Winter 2021] School of Engineering and Technology, University of Washington - Tacoma	L15.32

PROBABILISTIC FLOODING

- ….*Washington state in winter?*
- When a node is flooding a message, concept is to enforce a pro
- Throt
- Imple to achieve various p_{flood} scores
- With lower p_{flood} messages may not reach all nodes
- **USEFULNESS:** For random network with 10,000 nodes
- With p_{edge} = 0.1 and p_{flood} =.01
- Achieves 50-fold reduction in messages vs. full flooding

What does it mean to have p_{flood} =.01?

March 2, 2021	TCSS558: Applied Distributed Computing [Winter 2021] School of Engineering and Technology, University of Washington - Tacoma	L15.33

PROBABILISTIC FLOODING

- ….*Washington state in winter?*
- When a node is flooding a message, concept is to enforce a
- T
- I a
- W
- **USEFULNESS:** For random network with 10,000 nodes
- With p_{edge} = 0.1 and p_{flood} =.01
- Achieves 50-fold reduction in messages vs. full flooding

What does it mean to have p_{flood} =.01?

If a node Q has n neighbors, the probability that all neighbors don't forward the message to Q is $p=(1-p_{flood})^n$

March 2, 2021	TCSS558: Applied Distributed Computing [Winter 2021] School of Engineering and Technology, University of Washington - Tacoma	L15.34

PROBABILISTIC FLOODING

- ….*Washington state in winter?*
- W … rce a
- T
- I … a
- W
- U
- W
- Achieves 50-fold reduction in messages vs. full flooding

What does it mean to have p_{flood} =.01?

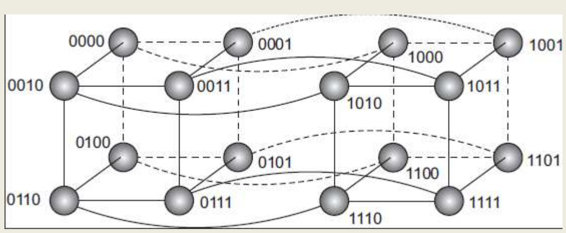
If a node Q has n neighbors, the probability that all neighbors don't forward the message to Q is $p=(1-p_{flood})^n$

if n=10, $p=(1-.01)^{10}=.904$ (pretty likely)
if n=100, $p=(1-.01)^{100}=.366$ (less likely)
if n=1000, $p=(1-.01)^{298}=.05$ (unlikely)

March 2, 2021	TCSS558: Applied Distributed Computing [Winter 2021] School of Engineering and Technology, University of Washington - Tacoma	L15.35

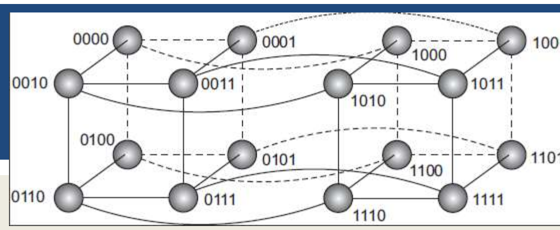
MESSAGE FLOODING

- For deterministic topologies (such as hypercube), design of efficient flooding scheme is much simpler
- If the overlay network is structured, this gives us a deterministic topology
- Schlosser et al [2002] – offer simple and efficient broadcasting scheme that relies on keeping track of neighbors per dimension



March 2, 2021	TCSS558: Applied Distributed Computing [Winter 2021] School of Engineering and Technology, University of Washington - Tacoma	L15.36

MESSAGE FLOODING - 2



- <u>**Hypercube Broadcast**</u>
- N(1001) starts the network broadcast
- N(1001) neighbors {0001,1000,1011,1101}
- N(1001) Sends message to all neighbors
- <u>**>>Edge Labels *(which bit is changed?, 1^{st}, 2^{nd}, 3^{rd}, 4^{th}…)***</u>
- Edge to 0001 – labeled 1 – change the 1st bit
- Edge to 1000 – labeled 4 – change the 4^{th} bit
- Edge to 1011 – labeled 3 – change the 3^{rd} bit
- Edge to 1101 – labeled 2 – change the 2^{nd} bit

- <u>**RULE: nodes only forward along edges with a higher dimension**</u>
- Node 1101 receives message on edge labeled 2
- Broadcast msg is only forwarded on <u>***higher***</u> valued edges (>2)

March 2, 2021	TCSS558: Applied Distributed Computing [Winter 2021] School of Engineering and Technology, University of Washington - Tacoma	L15.37

MESSAGE FLOODING - 3

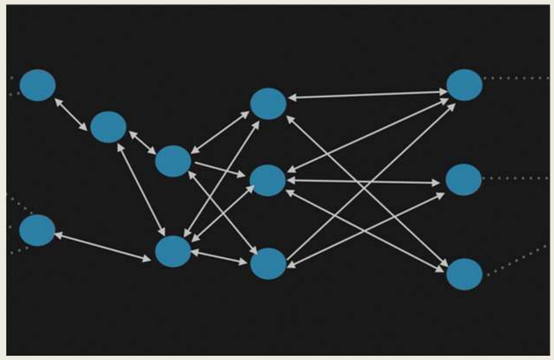
- <u>**Hypercube:**</u> forward msg along edges with higher dimension
- Node(1101)–neighbors {0101,1100,1001,1111}
- Node (1101) - incoming broadcast edge = 2
- <u>**Label Edges:**</u>
- Edge to 0101 – labeled 1 – change the 1^{st} bit
- Edge to 1100 – labeled 4 – change the 4^{th} bit <u>***<FORWARD>***</u>
- Edge to 1001 – labeled 2 – change the 2^{nd} bit
- Edge to 1111 – labeled 3 – change the 3^{rd} bit <u>***<FORWARD>***</u>
- N(1101) broadcast – forward only to N(1100) and N(1111)
- (1100) and (1111) are the <u>***higher dimension edges***</u>

- Broadcast requires just: N-1 messages, where nodes $N=2^n$, n=dimensions of hypercube

March 2, 2021	TCSS558: Applied Distributed Computing [Winter 2021] School of Engineering and Technology, University of Washington - Tacoma	L15.38

GOSSIP BASED DATA DISSEMINATION

- When structured peer-to-peer topologies are not available
- Gossip based approaches support multicast communication over unstructured peer-to-peer networks
- General approach is to leverage how gossip spreads across a group
- This is also called "epidemic behavior"…
- Data updates for a specific item begin at a specific node



INFORMATION DISSEMINATION

- **Epidemic algorithms**: algorithms for large-scale distributed systems that spread information
- Goal: "infect" all nodes with new information as fast as possible
- **Infected**: node with data that can spread to other nodes
- **Susceptible**: node without data
- **Removed**: node with data that is unable to spread data

EPIDEMIC PROTOCOLS

- **Gossiping**

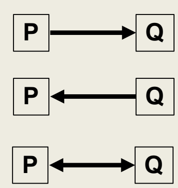
- Nodes are randomly selected

- One node, randomly selects any other node in the network to propagate the network

- Complete set of nodes is known to each member

ANTI ENTROPY DISSEMINATION MODEL FOR GOSSIPING

- **Anti-entropy:** Propagation model where node P picks node Q at random and exchanges message updates
- Akin to random walk



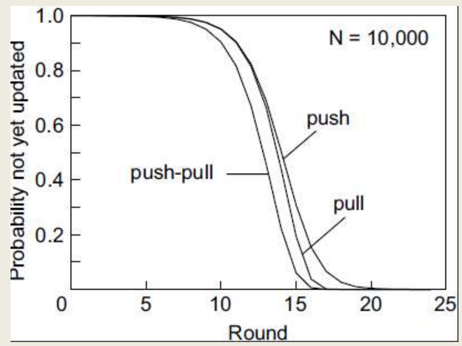
- **Types of message exchange:**
- **PUSH:** P only **pushes** its own updates to Q
- **PULL:** P only **pulls** in new updates from Q
- **TWO-WAY:** P and Q send updates to each other (i.e. a push-pull approach)

- Push only: hard to propagate updates to last few hidden susceptible nodes
- Pull: better because susceptible nodes can pull updates from infected nodes
- Push-pull is better still

ANTI ENTROPY EFFECTIVENESS

- **Round**: span of time during which every node takes initiative to exchange updates with a randomly chosen node
- The number of rounds to propagate a single update to all nodes requires O(log(N)), where N=number of nodes
- Let p_i denote probability that node P has not received msg m after the i^{th} round.
- For pull, push, and push-pull based approaches:

10,000 nodes →



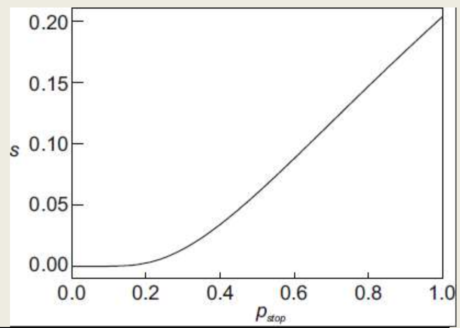
RUMOR SPREADING

- Variant of epidemic protocols
- Provides an approach to "stop" message spreading
- Mimics "gossiping" in real life

- **Rumor spreading:**
- **Node P** receives new data **item X**
- Contacts an arbitrary **node Q** to push update
- **Node Q** reports already receiving **item X** from another node
- **Node P** may loose interest in spreading the rumor with probability = p_{stop}, let's say 20% . . . (or 0.20)

RUMOR SPREADING - 2

- p_{stop}, is the probability node will stop spreading once contacting a node that already has the message
- Does not guarantee all nodes will be updated
- The fraction of nodes s, that remain susceptible grows relative to the probability that node P stops propagating when finding a node already having the message
- Fraction of nodes not updated remains < 0.20 with high p_{stop}
- Susceptible nodes (s) vs. probability of stopping →



REMOVING DATA

- Gossiping is good for spreading data
- <u>But how can data be removed from the system?</u>
- Idea is to issue ***"death certificates"***
- Act like data records, which are spread like data
- When death certificate is received, data is deleted
- Certificate is held to prevent data element from reinitializing from gossip from other nodes
- Death certificates time-out after expected time required for data element to clear out of entire system
- A few nodes maintain death certificates forever

DEATH CERTIFICATE EXAMPLE

- **For example:**
- **Node P** keeps death certificates forever
- **Item X** is removed from the system
- **Node P** receives an update request for **Item X**, but ***also*** holds the death certificate for **Item X**
- **Node P** will recirculate the death certificate across the network for **Item X**

WE WILL RETURN AT 2:46 PM

OBJECTIVES – 3/2

- Questions from 2/25
- Assignment 2: Replicated Key Value Store
- Chapter 4: Communication
  - Chapter 4.3: Message Oriented Communication
  - Chapter 4.4: Multicast Communication
- Chapter 6: Coordination
  - Chapter 6.1: Clock Synchronization

CHAPTER 6 - COORDINATION

- 6.1 Clock Synchronization
  - Physical clocks
  - Clock synchronization algorithms
- 6.2 Logical clocks
  - Lamport clocks
  - Vector clocks
- 6.3 Mutual exclusion
- 6.4 Election algorithms
- 6.6 Distributed event matching *(light)*
- 6.7 Gossip-based coordination *(light)*

CHAPTER 6 - COORDINATION

- How can processes synchronize and coordinate data?
- Process synchronization
  - Coordinate cooperation to grant individual processes temporary access to shared resources (e.g. a file)
- Data synchronization
  - Ensure two sets of data are the same (data replication)
- Coordination
  - Goal is to manage interactions and dependencies between activities in the distributed system
  - Encapsulates synchronization

COORDINATION - 2

- Synchronization challenges begin with <u>time</u>:
  - How can we synchronize computers, so they all agree on the time?
  - How do we measure and coordinate when things happen?
- Fortunately, for synchronization in distributed systems, it is often sufficient to only agree on a relative ordering of events
  - E.g. not actual time

COORDINATION - 3

- Groups of processes often appoint a **coordinator**
- **Election algorithms** can help elect a leader
- Synchronizing access to a shared resource is achieved with **distributed mutual exclusion** algorithms
- Also in chapter 6:
  - Matching subscriptions to publications in publish-subscribe systems
  - Gossip-based coordination problems:
    - Aggregation
    - Peer sampling
    - Overlay construction

OBJECTIVES – 3/2

- Questions from 2/25
- Assignment 2: Replicated Key Value Store
- Chapter 4: Communication
  - Chapter 4.3: Message Oriented Communication
  - Chapter 4.4: Multicast Communication
- Chapter 6: Coordination
  - **Chapter 6.1: Clock Synchronization**

CH. 6.1: CLOCK SYNCHRONIZATION

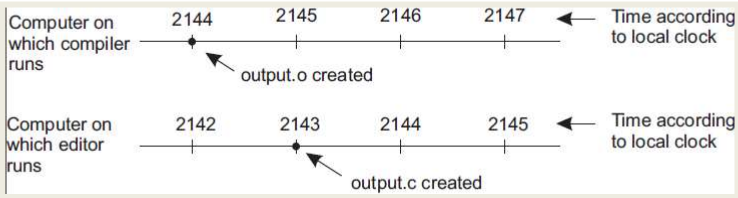
L15.55

CLOCK SYNCHRONIZATION

- Example:
- "make" is used to compile source files into binary object and executable files
- As an optimization, make only compiles files when the "last modified time" of source files is more recent than object and executables
- Consider if files are on a shared disk of a distributed system where there is no agreement on time
- Consider if the program has 1,000 source files

March 2, 2021	TCSS558: Applied Distributed Computing [Winter 2021] School of Engineering and Technology, University of Washington - Tacoma	L15.56

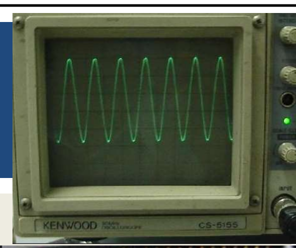
TIME SYNCHRONIZATION PROBLEM FOR DISTRIBUTED SYSTEMS



- Updates from different machines, may have clocks set to different times
- Make becomes confused with which files to recompile

March 2, 2021	TCSS558: Applied Distributed Computing [Winter 2021] School of Engineering and Technology, University of Washington - Tacoma	L15.57

PHYSICAL CLOCKS



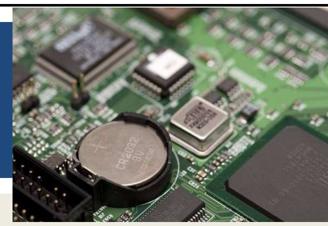
- Computer timers: precisely machined quartz crystals
- When under tension, they oscillate at a well defined frequency
- In analog electronics/communications crystals once used to set the frequency of two-way radio transceivers for
- Today, crystals are associated with a counter and holding register on a digital computer.

1960s ERA radio crystal →

- Each oscillation decrements a counter by one
- When counter gets to zero, an interrupt fires
- Can program timer to generate interrupt, let's say 60 times a second, or another frequency to track time

March 2, 2021	TCSS558: Applied Distributed Computing [Winter 2021] School of Engineering and Technology, University of Washington - Tacoma	L15.58

COMPUTER CLOCKS



- Digital clock on computer sets base time
- Crystal clock tracks forward progress of time
  - Translation of wave "ticks" to clock pulses
- CMOS battery on motherboard maintains clock on power loss

- <u>Clock skew</u>: physical clock crystals are not exactly the same
- Some run at slightly different rates
- Time differences accumulate as clocks drift forward or backward slightly

- In an automobile, where there is no clock synchronization, clock skew may become noticeable over months, years

March 2, 2021	TCSS558: Applied Distributed Computing [Winter 2021] School of Engineering and Technology, University of Washington - Tacoma	L15.59

UNIVERSAL COORDINATED TIME

- <u>Universal Coordinated Time (UTC)</u>

```
ubuntu@ip-172-31-58-89:~$ date
Thu Nov 16 10:13:39 UTC 2017
```

  - Worldwide standard for time keeping
  - Equivalent to Greenwich Mean Time (United Kingdom)
  - 40 shortwave radio stations around the world broadcast a short pulse at the start of each second (WWV)
  - World wide "atomic" clocks powered by constant transitions of the non-radioactive caesium-133 atom
    - 9,162,631,770 transitions per second

- Computers track time using UTC as a base
  - Avoid thinking in local time, which can lead to coordination issues
  - Operating systems may translate to show local time

March 2, 2021	TCSS558: Applied Distributed Computing [Winter 2021] School of Engineering and Technology, University of Washington - Tacoma	L15.60

COMPUTING: CLOCK CHALLENGES

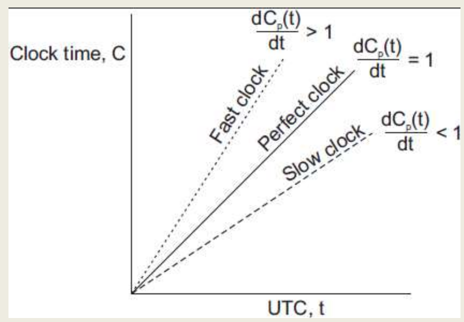
- How do we synchronize computer clocks with real-world clocks?
- How do we synchronize computer clocks with each other?

CLOCK SYNCHRONIZATION

- **UTC services**: use radio and satellite signals to provide time accuracy to 50ns
- **Time servers**: Server computers with UTC receivers that provide accurate time
- **Precision** (π): how close together a set of clocks may be
- **Accuracy**: how correct to actual time clocks may be
- **Internal synchronization**: Sync local computer clocks
- **External synchronization**: Sync to UTC clocks
- **Clock drift**: clocks on different machines gradually become out of sync due to crystal imperfections, temperature differences, etc.
- **Clock drift rate**: typical is 31.5s per year
- **Maximum clock drift rate** (ρ): clock specifications include one

CLOCK SYNCHRONIZATION - 2

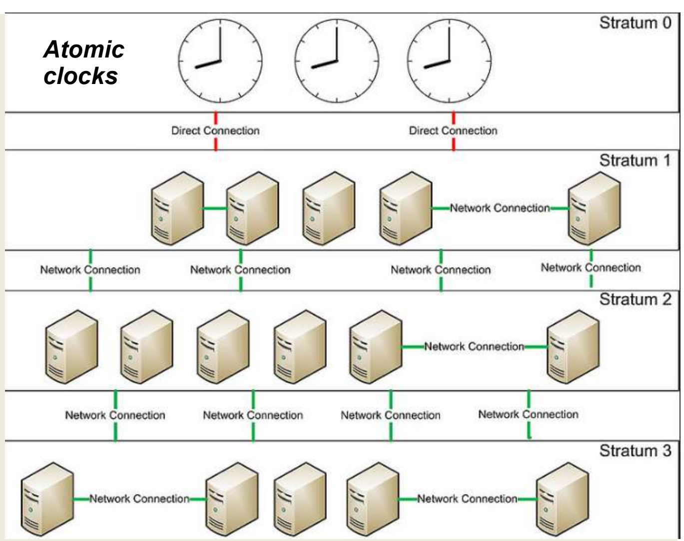
- If two clocks drift from UTC in opposite directions, after time Δt after synchronization, they may be 2ρ apart.
  - ρ - clock drift rate, π - clock precision (max 50ns)
- Clocks must be resynchronized every $\pi/2\rho$ seconds
- Network time protocol
- Provide coordination of time for servers
- Leverage distributed network of time servers



March 2, 2021	TCSS558: Applied Distributed Computing [Winter 2021] School of Engineering and Technology, University of Washington - Tacoma	L15.63

NETWORK TIME PROTOCOL

- Servers organized into stratums
- Stratum-1 servers have UTC receivers and are sync'd with atomic clocks
- Servers connect with closest NTP server for time synchronization
- Servers assume role as NTP server at stratum+1



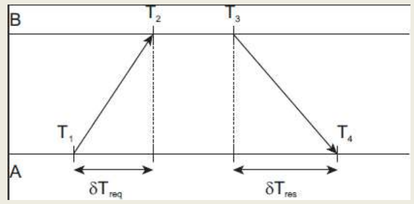
March 2, 2021	TCSS558: Applied Distributed Computing [Winter 2021] School of Engineering and Technology, University of Washington - Tacoma	L15.64

NTP - 2

- Must estimate network delays when synchronizing with remote UTC receiver clocks / time servers

Time server B

Client A



1. A sends message to B, with timestamp T1
2. B records time of receipt T2 (from local clock)
3. B returns response with send time T3, and receipt time T2
4. A records arrival of T4

- Assuming propagation delay of A→B→A is the same
- Estimate propagation delay: $\theta = T_3 + \frac{(T_2 - T_1) + (T_4 - T_3)}{2} - T_4 = \frac{(T_2 - T_1) + (T_3 - T_4)}{2}$
- Add delay to time

NTP - 3

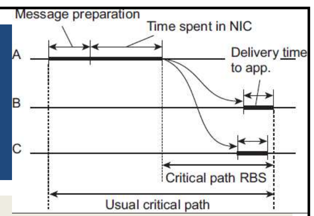
- Cannot set clocks backwards (recall "make" file example)
- Instead, temporarily slow the progress of time to allow fast clock to align with actual time
- Change rate of clock interrupt routine
- Slow progress of time until synchronized
- NTP accuracy is within 1-50ms

- In Ubuntu Linux, to quickly synchronize time:
  `$apt install ntp ntpdate`
- Specify local timeservers in /etc/ntp.conf
  ```
  server time.u.washington.edu iburst
  server bigben.cac.washington.edu iburst
  ```
- Shutdown service (sudo service ntp stop)
- Run ntpdate: (sudo ntpdate time.u.washington.edu)
- Startup service (sudo service ntp start)

BERKELEY ALGORITHM

- Berkeley time daemon server actively polls network to determine average time across servers
- Suitable when no machine has a UTC receiver
- Time daemon instructs servers how much to adjust clocks to achieve precision
- Accuracy can not be guaranteed
- Berkeley is an internal clock synchronization algorithm

CLOCK SYNCHRONIZATION IN WIRELESS NETWORKS



- Sensor networks bring unique challenges for clock synchronization
  - ***Address resource constraints***: limited power, multihop routing slow
- <u>**Reference broadcast synchronization (RBS)**</u>
- Provides precision of time, not accuracy as in Berkeley
- No UTC clock available
- RBS sender broadcasts a reference message to allow receivers to adjust clocks
- No multi-hop routing
- Time to propagate a signal to nodes is roughly constant
- Message propagation time does not consider time spent waiting in NIC for message to send
  - Wireless network resource contention may force wait before message even <u>***can***</u> be sent

REFERENCE BROADCAST SYNCHRONIZATION (RBS)

- Node broadcasts reference message m
- Each node p records time Tp,m when m is received
- Tp,m is read from node p's clock
- Two nodes p and q can exchange delivery times to estimate mutual relative offset
- Then calculate relative average offset for the network:

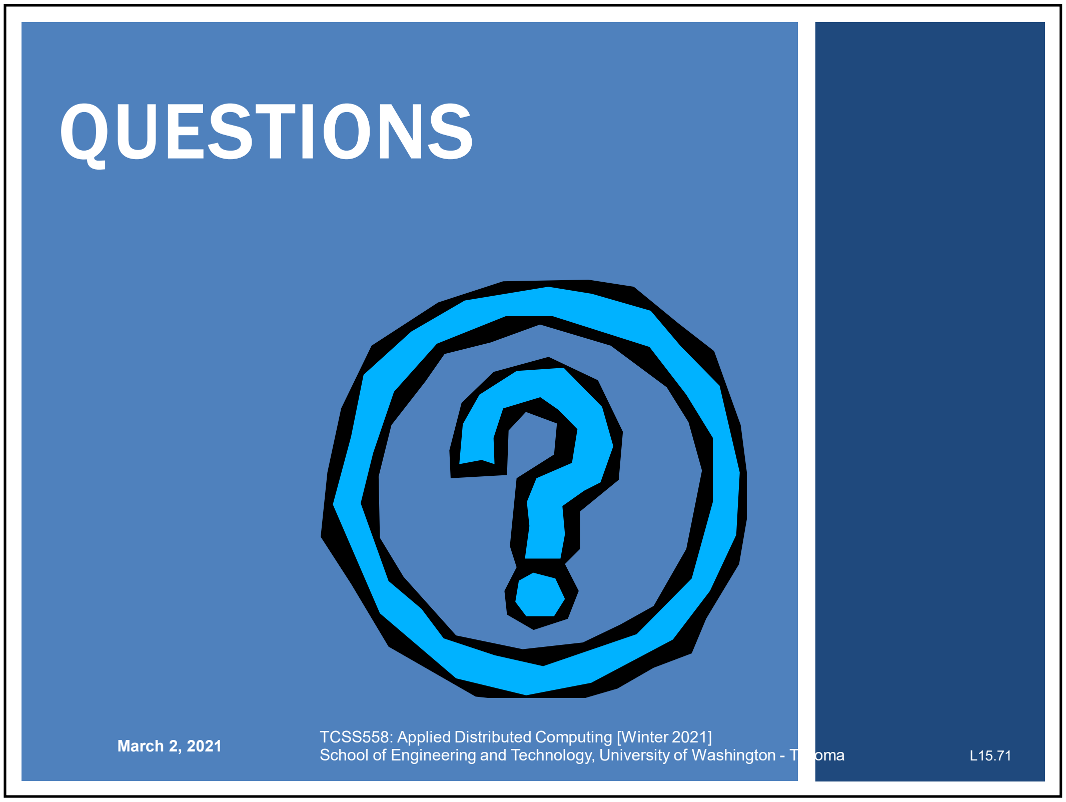
$$Offset[p,q] = \frac{\sum_{k=1}^{M}(T_{p,k} - T_{q,k})}{M}$$

- Where M is the total number of reference messages sent
- Nodes can simply store offsets instead of frequently synchronizing clocks to save energy

REFERENCE BROADCAST SYNCHRONIZATION (RBS) - 2

- Cloud skew: over time clocks drift apart
- Averages become less precise
- Elson et al. propose using standard linear regression to predict offsets, rather than calculating them
- IDEA: Use node's history of message times in a simple linear regression to continuously refine a formula with coefficients to predict time offsets:

$$Offset[p,q](t) = \alpha t + \beta$$


QUESTIONS
March 2, 2021
TCSS558: Applied Distributed Computing [Winter 2021]
School of Engineering and Technology, University of Washington -
L15.71