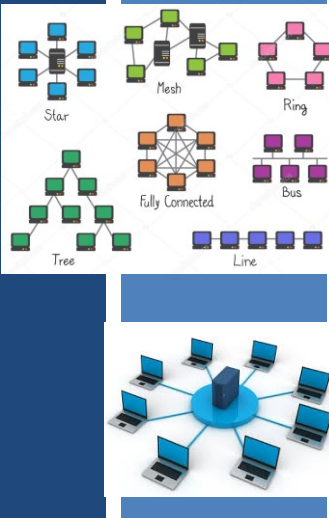


TCSS 558: APPLIED DISTRIBUTED COMPUTING

Chapter 4 - Communication-III
Chapter 6 - Coordination

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



The slide features a dark blue background with white text. On the right side, there is a grid of network topology diagrams: Star (a central node connected to multiple peripheral nodes), Mesh (a grid of interconnected nodes), Ring (nodes connected in a closed loop), Tree (a hierarchical structure of nodes), Fully Connected (every node connected to every other node), Bus (all nodes connected to a single central line), and Line (nodes connected in a straight line). Below these diagrams is a 3D illustration of a central server tower connected to several laptops arranged in a circle.

1

OBJECTIVES - 2/23

- **Questions from 2/21**
- 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

| | | |
|-------------------|---|-------|
| February 23, 2023 | TCSS558: Applied Distributed Computing [Winter 2023] School of Engineering and Technology, University of Washington - Tacoma | L15.2 |
|-------------------|---|-------|

2

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

TCSS 558 A > Assignments

Winter 2021

Home

Announcements

Assignments

Zoom

Chat

Search for Assignment

Upcoming Assignments

TCSS 558 - Online Daily Feedback Survey - 1/5
Not available until Jan 5 at 1:30pm | Due Jan 6 at 10pm | -1 pts

February 23, 2023 TCSS558: Applied Distributed Computing [Winter 2023] School of Engineering and Technology, University of Washington - Tacoma L15.3

3

TCSS 558 - Online Daily Feedback Survey - 1/5

Due Jan 6 at 10pm Points 1 Questions 4
Available Jan 5 at 1:30pm - Jan 6 at 11:59pm 1 day Time Limit None

Question 1 0.5 pts

On a scale of 1 to 10, please classify your perspective on material covered in today's class:

| | | | | | | | | | |
|------------------------|---|---|---|-------------------------|---|---|---|---|---------------------|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| Mostly Review To Me | | | | Equal New and Review | | | | | Mostly New to Me |

Question 2 0.5 pts

Please rate the pace of today's class:

| | | | | | | | | | |
|------|---|---|---|------------|---|---|---|---|------|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| Slow | | | | Just Right | | | | | Fast |

February 23, 2023 TCSS558: Applied Distributed Computing [Winter 2023] School of Engineering and Technology, University of Washington - Tacoma L15.4

4

MATERIAL / PACE

- Please classify your perspective on material covered in today's class (26 respondents):
 - 1-mostly review, 5-equal new/review, 10-mostly new
 - **Average - 6.42** (↑ - *previous 6.03*)
- Please rate the pace of today's class:
 - 1-slow, 5-just right, 10-fast
 - **Average - 5.81** (↑ - *previous 5.34*)

| | | |
|-------------------|---|-------|
| February 23, 2023 | TCSS558: Applied Distributed Computing [Winter 2023] School of Engineering and Technology, University of Washington - Tacoma | L15.5 |
|-------------------|---|-------|

5

FEEDBACK FROM 2/21

- *Is there a relationship between STUBS and Big-Endian?*
- Big-Endian notion refers to the ordering of bits in a byte stream that is exchanged over a network, or the ordering of bits for bytes stored in memory

• Typically data is transferred over the network using Big-endian notation

• Marshalling/Unmarshalling involves translating data from a machine-specific notion for exchange over the network

| | | |
|-------------------|---|-------|
| February 23, 2023 | TCSS558: Applied Distributed Computing [Winter 2023] School of Engineering and Technology, University of Washington - Tacoma | L15.6 |
|-------------------|---|-------|

6

FEEDBACK - 2

- STUBS refer to an interface definition that a client uses to invoke a remote procedure (RPC)
- **Under what circumstances will Multicast RPC be used?**
 - Execute multiple copies of a procedure – in case one fails
 - Reproduce/verify execution – second call verifies results of the first
 - Divide and conquer – individual RPC calls operate on a unique data subset
 - Race – execute procedure on multiple servers to obtain result faster

February 23, 2023

TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma

L15.7

7

OBJECTIVES - 2/23

- Questions from 2/21
- **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

February 23, 2023

TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma

L15.8

8

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 |

February 23, 2023

TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma

L15.9

9

ASSIGNMENT 2

- **Sunday March 12th**
- **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

February 23, 2023

TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma

L15.10

10

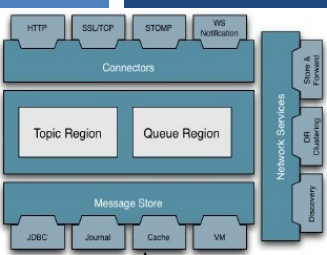
OBJECTIVES - 2/23

- Questions from 2/21
- 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

| | | |
|-------------------|---|--------|
| February 23, 2023 | TCSS558: Applied Distributed Computing [Winter 2023] School of Engineering and Technology, University of Washington - Tacoma | L15.11 |
|-------------------|---|--------|

11

CH. 4.3: MESSAGE-ORIENTED COMMUNICATION



The diagram illustrates the Apache ActiveMQ architecture. At the top, a 'Connectors' layer includes HTTP, SSL/TCP, STOMP, and WS Notification. Below this is the core messaging layer with 'Topic Region' and 'Queue Region'. The 'Message Store' layer includes JDBC, Journal, Cache, and VM. On the right, 'Network Services' include Store & Forward, UR, Discovery, and Discovery.

Apache ActiveMQ

L15.12

12

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”

| | | |
|-------------------|---|--------|
| February 23, 2023 | TCSS558: Applied Distributed Computing [Winter 2023] School of Engineering and Technology, University of Washington - Tacoma | L15.13 |
|-------------------|---|--------|

13

SOCKETS - 2

- Servers execute 1st - 4 operations (socket, bind, listen, accept)
- Methods refer to C API functions
- Mappings across different libraries will vary (e.g. Java)

| Operation | Description |
|-----------|---|
| socket | Create a new communication end point |
| bind | Attach local address to socket (IP / port) |
| listen | Tell OS what max # of pending connection requests should be |
| accept | Block caller until a connection request arrives |
| connect | Actively attempt to establish a connection |
| send | Send some data over the connection |
| receive | Receive some data over the connection |
| close | Release the connection |

| | | |
|-------------------|---|--------|
| February 23, 2023 | TCSS558: Applied Distributed Computing [Winter 2023] School of Engineering and Technology, University of Washington - Tacoma | L15.14 |
|-------------------|---|--------|

14

SERVER SOCKET OPERATIONS

- **Socket:** creates new communication end point
- **Bind:** associated IP and port with end point
- **Listen:** for connection-oriented communication, non-blocking call reserves buffers for specified number of pending connection requests server is willing to accept
- **Accept:** blocks until connection request arrives
 - Upon arrival, new socket is created matching original
 - Server spawns thread, or forks process to service incoming request
 - Server continues to wait for new connections on original socket

February 23, 2023

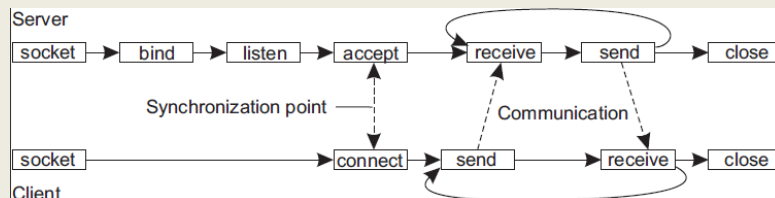
TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma

L15.15

15

CLIENT SOCKET OPERATIONS

- **Socket:** Creates socket client uses for communication
- **Connect:** Server transport-level address provided, client blocks until connection established
- **Send:** Supports sending data (to: server/client)
- **Receive:** Supports receiving data (from: server/client)
- **Close:** Closes communication channel
 - Analogous to closing a file stream



February 23, 2023

TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma

L15.16

16

SOCKET COMMUNICATION

- Sockets provide primitives for implementing your own TCP/UDP communication protocols
- Directly using sockets for transient (non-persisted) messaging is very basic, can be brittle
 - Easy to make mistakes...
- Any extra communication facilities must be implemented by the application developer
- More advanced approaches are desirable
 - E.g. frameworks with support common desirable functionality

February 23, 2023

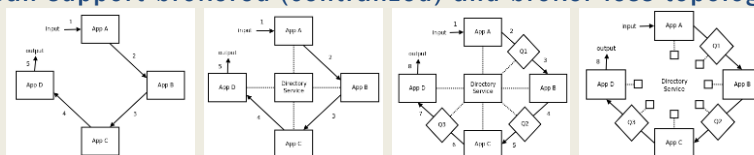
TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma

L15.17

17

ZEROMQ – SOCKET LIBRARY

- (0MQ) High performance intelligent **socket library**
- **zero broker, zero latency, zero admin, zero cost, zero waste**
- Provides a message queue
- **Builds upon** functionality of **traditional sockets**
- Implementation in C++
 - 30+ language bindings provided
- Enables support for various messaging patterns
- Can support brokered (centralized) and broker-less topologies



February 23, 2023

TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma

L15.18

18

ZEROMQ - 2

- ZeroMQ is **TCP-connection-oriented communication**
- Provides socket-like primitives with more functionality
 - Basic socket operations **abstracted** away
 - Supports many-to-one, one-to-one, and one-to-many connections
 - **Multicast** connections (one-to-many – single server socket simultaneously “connects” to multiple clients)
- Asynchronous messaging
- Supports pairing sockets to support **communication patterns**: *request-reply, pub-sub, pipe (queue)*

February 23, 2023

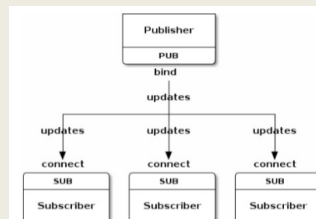
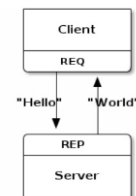
TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma

L15.19

19

ZEROMQ - PATTERNS

- **Request-reply pattern**
 - Traditional client-server communication (e.g. RPC)
 - Client: request socket (**REQ**)
 - Server: reply socket (**REP**)
- **Publish-subscribe pattern**
 - Clients **subscribe** to messages **published** by servers
 - As in event-based coordination (Ch. 1)
 - Supports multicasting messages from server to multiple subscribers (clients)
 - Client: subscribe socket (**SUB**)
 - Server: publish socket (**PUB**)



February 23, 2023

TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma

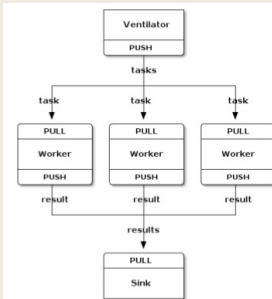
L15.20

20

ZEROMQ - PATTERNS - 2

■ Pipeline pattern (FIFO-queue)

- Analogous to a producer/consumer bounded buffer
- Producing processes generate results, push to pipe
- Consuming processes consume results, pull from pipe
- Producers: push socket (**PUSH** socket)
- Consumers: pull socket (**PULL** socket)
- Push- distributes messages to all pull clients evenly
- Consumers pull results from pipe and push results downstream



February 23, 2023

TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma

L15.21

21

QUEUEING ALTERNATIVES

- Cloud services
 - Amazon Simple Queueing Service (SQS)
 - Azure service bus
- Open source frameworks
 - Nanomsg
 - ZeroMQ
 - RabbitMQ

February 23, 2023

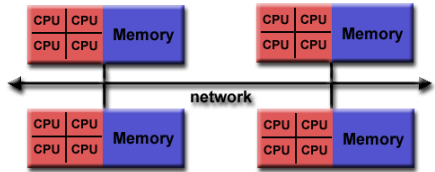
TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma

L15.22

22

MESSAGE PASSING INTERFACE (MPI)

- MPI introduced - version 1.0 March 1994
- Message passing API for parallel programming: supercomputers
- Communication protocol for parallel programming for:
Supercomputers, High Performance Computing (HPC) clusters
- Point-to-point and collective communication
- Goals: high performance, scalability, portability
- Most implementations
in C, C++, Fortran




The diagram shows four nodes arranged in a 2x2 grid. Each node is represented by a box containing four 'CPU' labels and one 'Memory' label. A double-headed arrow labeled 'network' connects the nodes horizontally and vertically, indicating communication paths.

| | | |
|-------------------|---|--------|
| February 23, 2023 | TCSS558: Applied Distributed Computing [Winter 2023] School of Engineering and Technology, University of Washington - Tacoma | L15.23 |
|-------------------|---|--------|

23

MOTIVATIONS FOR MPI

- Motivation: sockets insufficient for interprocess communication on large scale HPC compute clusters and super computers
 - Sockets at the wrong level of abstraction
 - Sockets designed to communicate over the network using general purpose TCP/IP stacks
 - Not designed for proprietary protocols
 - Not designed for high-speed interconnection networks used by supercomputers, HPC-clusters, etc.
 - Better buffering and synchronization needed



The image shows the cover of the book 'MPI-1 Message Passing Interface Standard, Version 1.1'. The cover is blue with white text.

| | | |
|-------------------|---|--------|
| February 23, 2023 | TCSS558: Applied Distributed Computing [Winter 2023] School of Engineering and Technology, University of Washington - Tacoma | L15.24 |
|-------------------|---|--------|

24

MOTIVATIONS FOR MPI - 2

- Supercomputers had proprietary communication libraries
 - Offer a wealth of efficient communication operations
- All libraries mutually incompatible (*vendor lock-in*)
- Led to significant portability problems developing parallel code that could migrate across supercomputers
- Led to development of MPI
 - To support transient (non-persistent) communication for parallel programming

February 23, 2023

TCSS558: Applied Distributed Computing [Winter 2023]
 School of Engineering and Technology, University of Washington - Tacoma

L15.25

25

MPI FUNCTIONS / DATATYPES

- Very large library, v1.0 (1994) 128 functions
- v3 (2015) 440+ functions
- MPI data types:
- Provide common mappings

| MPI datatype | C datatype |
|--------------------|--------------------|
| MPI.CHAR | signed char |
| MPI.SHORT | signed short int |
| MPI.INT | signed int |
| MPI.LONG | signed long int |
| MPI.UNSIGNED.CHAR | unsigned char |
| MPI.UNSIGNED.SHORT | unsigned short int |
| MPI.UNSIGNED.LONG | unsigned long int |
| MPI.FLOAT | float |
| MPI.DOUBLE | double |
| MPI.LONG.DOUBLE | long double |
| MPI.BYTE | |
| MPI.PACKED | |

| | | | |
|------------------------|---------------------------|---------------------------|-----------------------|
| MPI.ABORT | MPI.ADDRESS | MPI.ALLGATHER | MPI.ALLGATHERV |
| MPI.ALLREDUCE | MPI.ALLTOALL | MPI.ALLTOALLV | MPI.ATTR_DELETE |
| MPI.ATTR.GET | MPI.ATTR.PUT | MPI.BARRIER | MPI.BCAST |
| MPI.BSEND | MPI.BSEND_INIT | MPI.BUFFER.ATTACH | MPI.BUFFER.DETACH |
| MPI.CANCEL | MPI.CARTDIM.GET | MPI.CART.COORDS | MPI.CART.CREATE |
| MPI.CART.GET | MPI.CART.MAP | MPI.CART.RANK | MPI.CART.SHIFT |
| MPI.CART.SUB | MPI.COMM.COMPARE | MPI.COMM.CREATE | MPI.COMM.DUP |
| MPI.COMM.FREE | MPI.COMM.GROUP | MPI.COMM.RANK | MPI.COMM.REMOTE.GROUP |
| MPI.COMM.REMOTE.SIZE | MPI.COMM.SIZE | MPI.COMM.SPLIT | MPI.COMM.TEST.INTER |
| MPI.DIMS.CREATE | MPI.ERRHANDLER.CREATE | MPI.ERRHANDLER.FREE | MPI.ERRHANDLER.GET |
| MPI.ERRHANDLER.SET | MPI.ERROR.CLASS | MPI.ERROR.STRING | MPI.FINALIZE |
| MPI.GATHER | MPI.GATHERV | MPI.GET.COUNT | MPI.GET.ELEMENTS |
| MPI.GET.PROCESSOR.NAME | MPI.GRAPH.EDGES.GET | MPI.GRAPH.CREATE | MPI.GRAPH.GET |
| MPI.GRAPH.MAP | MPI.GRAPH.NEIGHBORS | MPI.GRAPH.NEIGHBORS.COUNT | MPI.GROUP.COMPARE |
| MPI.GROUP.DIFFERENCE | MPI.GROUP.EXCL | MPI.GROUP.FREE | MPI.GROUP.INCL |
| MPI.GROUP.INTERSECTION | MPI.GROUP.RANGE.EXCL | MPI.GROUP.RANGE.INCL | MPI.GROUP.RANK |
| MPI.GROUP.SIZE | MPI.GROUP.TRANSLATE.RANKS | MPI.GROUP.UNION | MPI.IBSEND |
| MPI.INIT | MPI.INITIALIZED | MPI.INTERCOMM.CREATE | MPI.INTERCOMM.MERGE |
| MPI.IPROBE | MPI.IRCV | MPI.ISEND | MPI.ISEND |
| MPI.ISSEND | MPI.KEYVAL.CREATE | MPI.KEYVAL.FREE | MPI.OP.CREATE |
| MPI.OP.FREE | MPI.PACK | MPI.PACK.SIZE | MPI.PCONTROL |
| MPI.PROBE | MPI.RECV | MPI.RECV.INIT | MPI.REDUCE |
| MPI.REDUCE.SCATTER | MPI.REQUEST.FREE | MPI.RSEND | MPI.RSEND.INIT |
| MPI.SCAN | MPI.SCATTER | MPI.SCATTERV | MPI.SEND |
| MPI.SENDRECV | MPI.SENDRECV.REPLACE | MPI.SEND.INIT | MPI.SSEND |
| MPI.SSEND.INIT | MPI.START | MPI.STARTALL | MPI.TEST |
| MPI.TESTALL | MPI.TESTANY | MPI.TESTSOME | MPI.TEST.CANCELLED |
| MPI.TOPO.TEST | MPI.TYPE.COMMIT | MPI.TYPE.CONTIGUOUS | MPI.TYPE.EXTENT |
| MPI.TYPE.FREE | MPI.TYPE.HINDEXED | MPI.TYPE.IVECTOR | MPI.TYPE.INDEXED |
| MPI.TYPE.LB | MPI.TYPE.SIZE | MPI.TYPE.STRUCT | MPI.TYPE.LB |
| MPI.TYPE.VECTOR | MPI.UNPACK | MPI.WAIT | MPI.WAITALL |
| MPI.WALTANY | MPI.WAITSONE | MPI.WTICK | MPI.WTIME |

February 23, 2023

TCSS558: Applied Distributed Computing [Winter 2023]
 School of Engineering and Technology, University of Washington - Tacoma

L15.26

26

COMMON MPI FUNCTIONS

- MPI - no recovery for process crashes, network partitions
- Communication among grouped processes: $(groupID, processID)$
- IDs used to route messages in place of IP addresses

| Operation | Description |
|--------------|---|
| MPI_bsend | Append outgoing message to a local send buffer |
| MPI_send | Send message, wait until copied to local/remote buffer |
| MPI_ssend | Send message, wait until transmission starts |
| MPI_sendrecv | Send message, wait for reply |
| MPI_issend | Pass reference to outgoing message and continue |
| MPI_issend | Pass reference to outgoing messages, wait until receipt start |
| MPI_recv | Receive a message, block if there is none |
| MPI_irecv | Check for incoming message, do not block! |

February 23, 2023

TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma

L15.27

27

MESSAGE-ORIENTED-MIDDLEWARE

- **Message-queueing systems**
 - Provide extensive support for *persistent* asynchronous communication
 - In contrast to transient systems
 - *Temporally decoupled*: messages are eventually delivered to recipient queues
- Message transfers may take minutes vs. sec or ms
- Each application has its own private queue to which other applications can send messages

February 23, 2023

TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma

L15.28

28

MESSAGE QUEUEING SYSTEMS: USE CASES

- Enables communication between applications, or sets of processes
 - User applications
 - App-to-database
 - To support distributed real-time computations

- Use cases
 - Batch processing, Email, workflow, groupware, routing subqueries

February 23, 2023
TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
L15.29

29

MESSAGE QUEUEING SYSTEMS

- Scenarios:
 - (a) Sender/receiver both running
 - (b) Sender running, receiver offline
 - (c) Sender offline, receiver running
 - (d) Sender/receiver both offline
- Queue persists msgs, and attempts to send them but no one may be available to receive them...

| Sender running | Sender running | Sender passive | Sender passive |
|------------------|------------------|------------------|------------------|
| | | | |
| | | | |
| | | | |
| Receiver running | Receiver passive | Receiver running | Receiver passive |
| (a) | (b) | (c) | (d) |

February 23, 2023
TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma
L15.30

30

MESSAGE QUEUEING SYSTEMS - 2

- **Key:** Truly persistent messaging
- Message queueing systems can persist messages for awhile and senders and receivers can be offline
- **Messages**
 - Contain any data, may have size limit
 - Are properly addressed, to a destination queue
- **Basic Interface**
 - **PUT:** called by sender to append msg to specified queue
 - **GET:** blocking call to remove oldest msg from specified queue
 - Blocked if queue is empty
 - **POLL:** Non-blocking, gets msg from specified queue

February 23, 2023

TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma

L15.31

31

MESSAGE QUEUEING SYSTEMS ARCHITECTURE

- **Basic Interface cont'd**
- **NOTIFY:** install a callback function, for when msg is placed into a queue. Notifies receivers
- **Queue managers:** manage individual message queues as a separate process/library
- Applications get/put messages only from **local** queues
- Queue manager and apps share local network
- **ISSUES:**
 - How should we reference the destination queue?
 - How should names be resolved (looked-up)?
 - Contact address (host, port) pairs
 - Local look-up tables can be stored at each queue manager

February 23, 2023

TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma

L15.32

32

MESSAGE QUEUEING SYSTEMS ARCHITECTURE - 2

- **ISSUES:**
 - How do we route traffic between queue managers?
 - How are name-to-address mappings efficiently kept?
 - Each queue manager should be known to all others
- **Message brokers**
 - Handle message conversion among different users/formats
 - Addresses cases when senders and receivers don't speak the same protocol (language)
 - Need arises for message protocol converters
 - "Reformatter" of messages
 - Act as application-level gateway

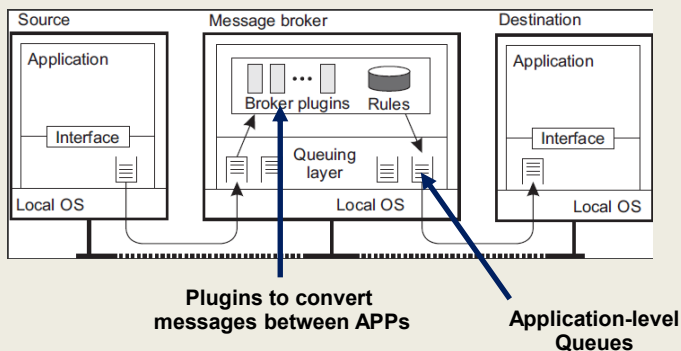
February 23, 2023

TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma

L15.33

33

MESSAGE BROKER ORGANIZATION



February 23, 2023

TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma

L15.34

34

AMQP PROTOCOL

- Message-queueing systems initially developed to enable legacy applications to interoperate
- Decouple inter-application communication using “open” messaging-middleware
- Many msg-queues are proprietary solutions, **so not very open**
- e.g. Microsoft Message Queueing service, Windows NT 1997
- **Advanced message queuing 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+

February 23, 2023

TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma

L15.35

35

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

February 23, 2023

TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma

L15.36

36

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)

February 23, 2023

TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma

L15.37

37

MESSAGE-ORIENTED-MIDDLEWARE EXAMPLES:

- **Some examples:**
- RabbitMQ, Apache QPid
 - Implement Advanced Message Queueing Protocol (AMQP)
- Apache Kafka – more of a pub/sub than plain msg queue
 - **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

February 23, 2023

TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma

L15.38

38

OBJECTIVES - 2/23

- Questions from 2/21
- 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

| | | |
|-------------------|---|--------|
| February 23, 2023 | TCSS558: Applied Distributed Computing [Winter 2023] School of Engineering and Technology, University of Washington - Tacoma | L15.39 |
|-------------------|---|--------|

39

CH. 4.4: MULTICAST COMMUNICATION

Multicast

one to many
X = subscriber

Apache ActiveMQ

L15.40

40

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”

| | | |
|-------------------|---|--------|
| February 23, 2023 | TCSS558: Applied Distributed Computing [Winter 2023] School of Engineering and Technology, University of Washington - Tacoma | L15.41 |
|-------------------|---|--------|

41

MULTICAST COMMUNICATION

- Sending data to multiple receivers (*one to many*)
- 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

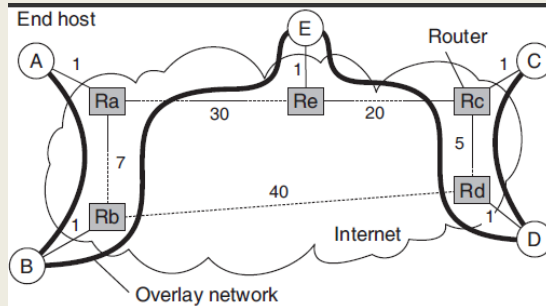
- Focus shifted more recently to **peer-to-peer** networks
 - **Structured overlay networks** can be setup easily and provide efficient communication paths
 - Application-level multicasting techniques more successful
 - Gossip-based dissemination: unstructured p2p networks

| | | |
|-------------------|---|--------|
| February 23, 2023 | TCSS558: Applied Distributed Computing [Winter 2023] School of Engineering and Technology, University of Washington - Tacoma | L15.42 |
|-------------------|---|--------|

42

NETWORK STRUCTURE

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



February 23, 2023

TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma

L15.43

43

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

February 23, 2023

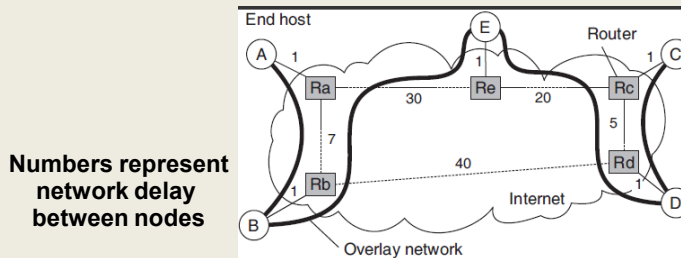
TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma

L15.44

44

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



February 23, 2023

TCSS558: Applied Distributed Computing [Winter 2023]
 School of Engineering and Technology, University of Washington - Tacoma

L15.45

45

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

February 23, 2023

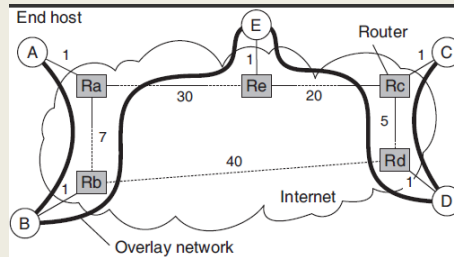
TCSS558: Applied Distributed Computing [Winter 2023]
 School of Engineering and Technology, University of Washington - Tacoma

L15.46

46

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?

February 23, 2023

TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma

L15.47

47

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

February 23, 2023

TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma

L15.48

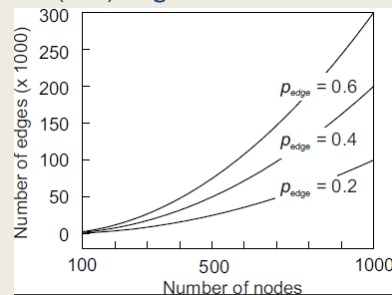
48

RANDOM GRAPHS

- Used when 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: $\frac{1}{2} * P_{\text{edge}} * N * (N-1)$ 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...



February 23, 2023

TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma

L15.49

49

PROBABILISTIC FLOODING



Washington state in winter?

- When a node is flooding a message, the concept is to enforce a probability that the message is spread (p_{flood})
- Throttle message flooding based on a probability
- Implementation needs to consider # 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 $p_{\text{edge}} = 0.1$ and $p_{\text{flood}} = 0.01$
- Achieves 50-fold reduction in messages vs. full flooding


February 23, 2023

TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma

L15.50

50

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})
- Throttling
- Implementation: How many edges does network with 10,000 nodes have with $p_{\text{edge}}=0.1$? to achieve
- With $p_{\text{edge}}=0.1$ and $p_{\text{flood}}=0.01$
- USEFULNESS:** For random network with 10,000 nodes
 - With $p_{\text{edge}} = 0.1$ and $p_{\text{flood}} = 0.01$
 - Achieves 50-fold reduction in messages vs. full flooding

| | | |
|-------------------|---|--------|
| February 23, 2023 | TCSS558: Applied Distributed Computing [Winter 2023] School of Engineering and Technology, University of Washington - Tacoma | L15.51 |
|-------------------|---|--------|

51

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})
- Throttling
- Implementation: How many edges does network with 10,000 nodes have with $p_{\text{edge}}=0.1$? to achieve
- With $p_{\text{edge}}=0.1$ and $p_{\text{flood}}=0.01$
Edges = $\frac{1}{2} * P_{\text{edge}} * N * (N-1)$
- USEFULNESS:** For random network with 10,000 nodes
 - With $p_{\text{edge}} = 0.1$ and $p_{\text{flood}} = 0.01$
 - Achieves 50-fold reduction in messages vs. full flooding

| | | |
|-------------------|---|--------|
| February 23, 2023 | TCSS558: Applied Distributed Computing [Winter 2023] School of Engineering and Technology, University of Washington - Tacoma | L15.52 |
|-------------------|---|--------|

52

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})
- Throt
- Imple **How many edges does network with 10,000 nodes have with $p_{\text{edge}}=0.1$?** to
- achieve
- With **Edges = $\frac{1}{2} * P_{\text{edge}} * N * (N-1)$**
- USE** $\frac{1}{2} * (.1) * (10000) * (9999)$ es
- With
- Achieves 50-fold reduction in messages vs. full flooding

| | | |
|-------------------|---|--------|
| February 23, 2023 | TCSS558: Applied Distributed Computing [Winter 2023] School of Engineering and Technology, University of Washington - Tacoma | L15.53 |
|-------------------|---|--------|

53

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})
- Throt
- Imple **How many edges does network with 10,000 nodes have with $p_{\text{edge}}=0.1$?** to
- achieve
- With **Edges = $\frac{1}{2} * P_{\text{edge}} * N * (N-1)$**
- USE** $\frac{1}{2} * (.1) * (10000) * (9999)$ es
- With **4,999,500 edges**
- Achieves 50-fold reduction in messages vs. full flooding

| | | |
|-------------------|---|--------|
| February 23, 2023 | TCSS558: Applied Distributed Computing [Winter 2023] School of Engineering and Technology, University of Washington - Tacoma | L15.54 |
|-------------------|---|--------|

54

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})
- Throt
- Imple
- achie
- With lower p_{flood} messages may not reach all nodes
- USEFULNESS:** For random network with 10,000 nodes
- With $p_{\text{edge}} = 0.1$ and $p_{\text{flood}} = .01$
- Achieves 50-fold reduction in messages vs. full flooding

| | | |
|-------------------|---|--------|
| February 23, 2023 | TCSS558: Applied Distributed Computing [Winter 2023] School of Engineering and Technology, University of Washington - Tacoma | L15.55 |
|-------------------|---|--------|

55

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})
- T
- I
- a
- W
- u
- With $p_{\text{edge}} = 0.1$ and $p_{\text{flood}} = .01$
- Achieves 50-fold reduction in messages vs. full flooding


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

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

| | | |
|-------------------|---|--------|
| February 23, 2023 | TCSS558: Applied Distributed Computing [Winter 2023] School of Engineering and Technology, University of Washington - Tacoma | L15.56 |
|-------------------|---|--------|

56

PROBABILISTIC FLOODING



Washington state in winter?

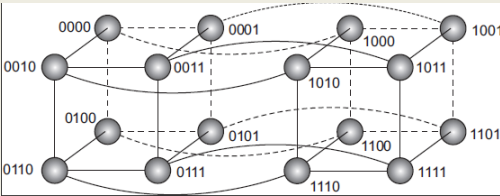
- **What does it mean to have $p_{\text{flood}} = .01$?**
- **If a node Q has n neighbors, the probability that all neighbors don't forward the message to Q is $p = (1 - p_{\text{flood}})^n$**
- **if $n=10$, $p = (1 - .01)^{10} = .904$ (pretty likely)**
- **if $n=100$, $p = (1 - .01)^{100} = .366$ (less likely)**
- **if $n=298$, $p = (1 - .01)^{298} = .05$ (unlikely)**

| | | |
|-------------------|---|--------|
| February 23, 2023 | TCSS558: Applied Distributed Computing [Winter 2023] School of Engineering and Technology, University of Washington - Tacoma | L15.57 |
|-------------------|---|--------|

57

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**



| | | |
|-------------------|---|--------|
| February 23, 2023 | TCSS558: Applied Distributed Computing [Winter 2023] School of Engineering and Technology, University of Washington - Tacoma | L15.58 |
|-------------------|---|--------|

58

MESSAGE FLOODING - 2

- **Hypercube Broadcast**
- N(1001) starts the network broadcast
- N(1001) neighbors {0001,1000,1011,1101}
- N(1001) Sends message to all neighbors
- **>>Edge Labels (which bit is changed?, 1st, 2nd, 3rd, 4th...)**
- Edge to 0001 - labeled 1 - change the 1st bit
- Edge to 1000 - labeled 4 - change the 4th bit
- Edge to 1011 - labeled 3 - change the 3rd bit
- Edge to 1101 - labeled 2 - change the 2nd bit
- **RULE: nodes only forward along edges with a higher dimension**
- Node 1101 receives message on edge labeled 2
- Broadcast msg is only forwarded on **higher** valued edges (>2)

| | | |
|-------------------|---|--------|
| February 23, 2023 | TCSS558: Applied Distributed Computing [Winter 2023] School of Engineering and Technology, University of Washington - Tacoma | L15.59 |
|-------------------|---|--------|

59

MESSAGE FLOODING - 3

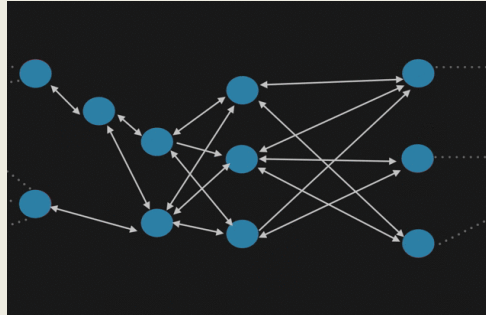
- **Hypercube:** forward msg along edges with higher dimension
- Node(1101)-neighbors {0101,1100,1001,1111}
- Node (1101) - incoming broadcast edge = 2
- **Label Edges:**
- Edge to 0101 - labeled 1 - change the 1st bit
- Edge to 1100 - labeled 4 - change the 4th bit ***<FORWARD>***
- Edge to 1001 - labeled 2 - change the 2nd bit
- Edge to 1111 - labeled 3 - change the 3rd bit ***<FORWARD>***
- N(1101) broadcast - forward only to N(1100) and N(1111)
- (1100) and (1111) are the **higher dimension edges**
- Broadcast requires just: N-1 messages, where nodes $N=2^n$, n =dimensions of hypercube

| | | |
|-------------------|---|--------|
| February 23, 2023 | TCSS558: Applied Distributed Computing [Winter 2023] School of Engineering and Technology, University of Washington - Tacoma | L15.60 |
|-------------------|---|--------|

60

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



February 23, 2023

TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma

L15.61

61

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

February 23, 2023

TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma

L15.62

62

EPIDEMIC PROTOCOLS

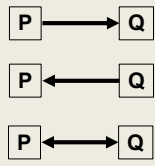
- **Gossiping**
- Nodes are randomly selected
- One node, randomly selects any other node in the network to propagate the network
- Complete set of nodes is known to each member

| | | |
|-------------------|---|--------|
| February 23, 2023 | TCSS558: Applied Distributed Computing [Winter 2023] School of Engineering and Technology, University of Washington - Tacoma | L15.63 |
|-------------------|---|--------|

63

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



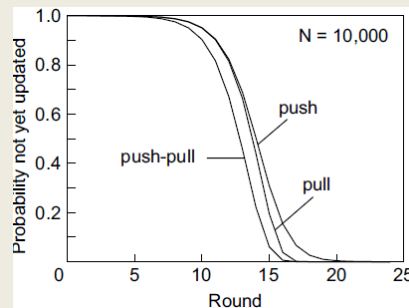
| | | |
|-------------------|---|--------|
| February 23, 2023 | TCSS558: Applied Distributed Computing [Winter 2023] School of Engineering and Technology, University of Washington - Tacoma | L15.64 |
|-------------------|---|--------|

64

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 →



February 23, 2023

TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma

L15.65

65

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)

February 23, 2023

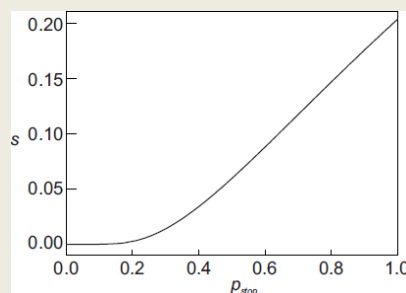
TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma

L15.66

66

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 →



February 23, 2023

TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma

L15.67

67

REMOVING DATA

- Gossiping is good for spreading data
- But how can data be removed from the system?
- 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

February 23, 2023

TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma

L15.68

68

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**

| | | |
|-------------------|---|--------|
| February 23, 2023 | TCSS558: Applied Distributed Computing [Winter 2023] School of Engineering and Technology, University of Washington - Tacoma | L15.69 |
|-------------------|---|--------|

69

WE WILL RETURN AT 3:02 PM



70

OBJECTIVES - 2/23

- Questions from 2/21
- 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

| | | |
|-------------------|---|--------|
| February 23, 2023 | TCSS558: Applied Distributed Computing [Winter 2023] School of Engineering and Technology, University of Washington - Tacoma | L15.71 |
|-------------------|---|--------|

71

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*)

| | | |
|-------------------|---|--------|
| February 23, 2023 | TCSS558: Applied Distributed Computing [Winter 2023] School of Engineering and Technology, University of Washington - Tacoma | L15.72 |
|-------------------|---|--------|

72

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

February 23, 2023

TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma

L15.73

73

COORDINATION - 2

- Synchronization challenges begin with time:
 - 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

February 23, 2023

TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma

L15.74

74

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

February 23, 2023

TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma

L15.75

75

OBJECTIVES - 2/23

- Questions from 2/21
- 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**

February 23, 2023

TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma

L15.76

76



CH. 6.1: CLOCK SYNCHRONIZATION

L15.77

77

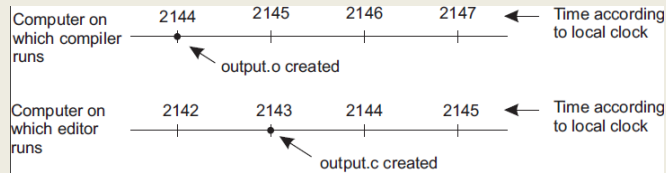
CLOCK SYNCHRONIZATION

- **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

| | | |
|-------------------|---|--------|
| February 23, 2023 | TCSS558: Applied Distributed Computing [Winter 2023] School of Engineering and Technology, University of Washington - Tacoma | L15.78 |
|-------------------|---|--------|

78

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

February 23, 2023

TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma

L15.79

79

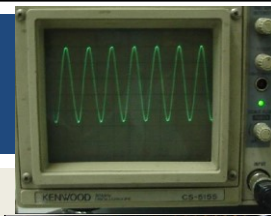


PHYSICAL CLOCKS

- **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



February 23, 2023



TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma

L15.80

80

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
- **Clock skew**: 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



| | | |
|-------------------|---|--------|
| February 23, 2023 | TCSS558: Applied Distributed Computing [Winter 2023] School of Engineering and Technology, University of Washington - Tacoma | L15.81 |
|-------------------|---|--------|

81

UNIVERSAL COORDINATED TIME

- **Universal Coordinated Time (UTC)** `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

| | | |
|-------------------|---|--------|
| February 23, 2023 | TCSS558: Applied Distributed Computing [Winter 2023] School of Engineering and Technology, University of Washington - Tacoma | L15.82 |
|-------------------|---|--------|

82

COMPUTING: CLOCK CHALLENGES

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

February 23, 2023

TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma

L15.83

83

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

February 23, 2023

TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma

L15.84

84

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

| | | |
|-------------------|---|--------|
| February 23, 2023 | TCSS558: Applied Distributed Computing [Winter 2023] School of Engineering and Technology, University of Washington - Tacoma | L15.85 |
|-------------------|---|--------|

85

NETWORK TIME PROTOCOL

- Servers organized into stratum
- 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

| | | |
|-------------------|---|--------|
| February 23, 2023 | TCSS558: Applied Distributed Computing [Winter 2023] School of Engineering and Technology, University of Washington - Tacoma | L15.86 |
|-------------------|---|--------|

86

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

| | | |
|-------------------|---|--------|
| February 23, 2023 | TCSS558: Applied Distributed Computing [Winter 2023] School of Engineering and Technology, University of Washington - Tacoma | L15.87 |
|-------------------|---|--------|

87

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)

| | | |
|-------------------|---|--------|
| February 23, 2023 | TCSS558: Applied Distributed Computing [Winter 2023] School of Engineering and Technology, University of Washington - Tacoma | L15.88 |
|-------------------|---|--------|

88

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

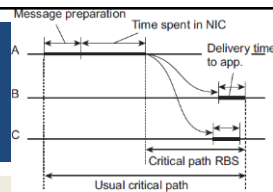
February 23, 2023

TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma

L15.89

89

CLOCK SYNCHRONIZATION IN WIRELESS NETWORKS



- Sensor networks bring unique challenges for clock synchronization
 - **Address resource constraints:** limited power, multihop routing slow
- **Reference broadcast synchronization (RBS)**
- 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 **can** be sent

February 23, 2023

TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma

L15.90

90

REFERENCE BROADCAST SYNCHRONIZATION (RBS)

- Node broadcasts reference message m
- Each node p records time $T_{p,m}$ when m is received
- $T_{p,m}$ 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:

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

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

February 23, 2023

TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma

L15.91

91

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:

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

February 23, 2023

TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma

L15.92

92

QUESTIONS

February 23, 2023

TCSS558: Applied Distributed Computing [Winter 2023]
School of Engineering and Technology, University of Washington - Tacoma

L15.93

93