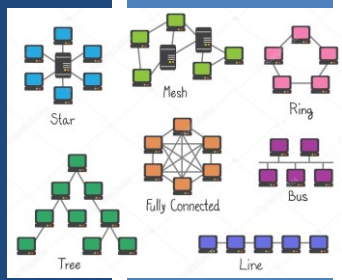
TCSS 558: APPLIED DISTRIBUTED COMPUTING



Ch. 4 – Communication - II

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

1

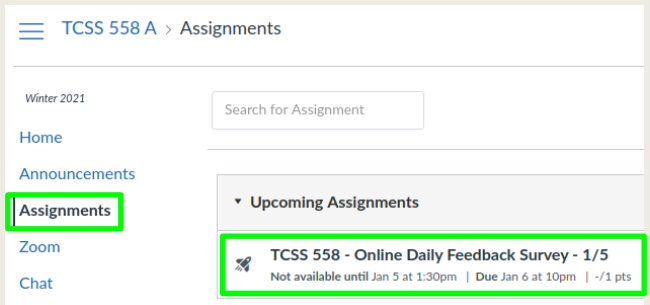
OBJECTIVES – 2/20

- Questions from 2/15
- Assignment 2: Key Value Store
- Assignment 3: Replicated Key Value Store
- Chapter 4: Communication
  - Chapter 4.1: Foundations
  - Chapter 4.2: RPC (light-overview)
  - Chapter 4.3: Message Oriented Communication
  - Chapter 4.4: Multicast Communication

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



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

4

MATERIAL / PACE

- Please classify your perspective on material covered in today's class (24 respondents):
- 1-mostly review, 5-equal new/review, 10-mostly new
- **Average – 6.17 (↑ - *previous 6.16*)**

- Please rate the pace of today's class:
- 1-slow, 5-just right, 10-fast
- **Average – 5.75 (↑ - *previous 5.29*)**

5

CSS TENURE TRACK FACULTY CANDIDATE RESEARCH SEMINARS – EXTRA CREDIT

- **Week 8:**
  - Tuesday February 20 – 12:30pm –KEY 102
  - Thursday February 22 – 12:30pm –MLG 110
  - Friday February 23 – 12:30pm –MLG 301
- **Week 9 (every day):**
  - Monday February 26 – 12:30pm – MLG 110
  - Wednesday February 28 – 1:30pm – JOY 117
  - Thursday February 29 – 1:30pm – MLG 110
  - Friday March 1 – 1:30pm – MLG 301
- **Week 10 (Monday and Tuesday):**
  - Mar 4, 5, 7 – 1:30pm – room TBA

In Winter 23, the final exam scores ran approx. 10 pts on average below the W'24 midterm scores

- Earn up to 30 buffer points added to the Final Exam score
- Earn 3 points for each seminar attended
- Buffer points replace missed points on the Final Exam
- Once the Final Exam score = 100%, additional points do not push the Final Exam score above 100%
- Buffer points will not impact the course curve for the Final Exam
- Any course curve will be applied before buffer points

6

FEEDBACK FROM 2/15

7

OBJECTIVES – 2/20

- Questions from 2/15
- **Assignment 2: Key Value Store**
- Assignment 3: Replicated Key Value Store
- Chapter 4: Communication
  - Chapter 4.1: Foundations
  - Chapter 4.2: RPC (light-overview)
  - Chapter 4.3: Message Oriented Communication
  - Chapter 4.4: Multicast Communication

8

ASSIGNMENT 2

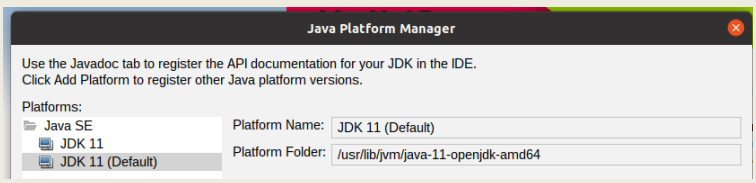
- **Find Teammates**: signup posted on Canvas under 'People'
- GenericNode.tar.gz includes Dockerfile examples
- GenericNode.tar.gz assumes Java 11
- TCP/UDP/RMI Key Value Store
- Implement a "GenericNode" project which assumes the role of a client or server for a Key/Value Store
- Recommended in Java 11 LTS
- Client node program interacts with server node to put, get, delete, or list items in a key/value store

February 20, 2024	TCSS558: Applied Distributed Computing [Winter 2024] School of Engineering and Technology, University of Washington - Tacoma	L13.9

9

USING JAVA 11 IN NETBEANS

- In Netbeans IDE, under Tools menu, 'Java Platforms', be sure to install and select JDK 11



- On left-hand Project menu, right-click on '**GenericNode**' project
- Select Properties
- Under Build | Compile, be sure Java Platform is JDK 11
- Under Sources, be sure Source/Binary Format is 11

February 20, 2024	TCSS558: Applied Distributed Computing [Winter 2024] School of Engineering and Technology, University of Washington - Tacoma	L13.10

10

OBJECTIVES – 2/20

- Questions from 2/15
- Assignment 2: Key Value Store
- Assignment 3: Replicated Key Value Store
- Chapter 4: Communication
  - Chapter 4.1: Foundations
  - Chapter 4.2: RPC (light-overview)
  - Chapter 4.3: Message Oriented Communication
  - Chapter 4.4: Multicast Communication

11

ASSIGNMENT 3 – COMING SOON

- DUE Sunday March 10th
- Goal: Replicated Key Value Store
- Team signup to be posted on Canvas under 'People'
- Builds off of Assignment 2 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

12

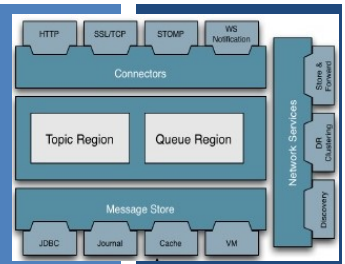
OBJECTIVES – 2/20

- Questions from 2/15
- Assignment 2: Key Value Store
- Assignment 3: Replicated Key Value Store
- Chapter 4: Communication
  - Chapter 4.1: Foundations
  - Chapter 4.2: RPC (light-overview)
  - **Chapter 4.3: Message Oriented Communication**
  - Chapter 4.4: Multicast Communication

February 20, 2024	TCSS558: Applied Distributed Computing [Winter 2024] School of Engineering and Technology, University of Washington - Tacoma	L13.13

13

CH. 4.3: MESSAGE-ORIENTED COMMUNICATION



Apache ActiveMQ

L13.14

14

CHAPTER 4

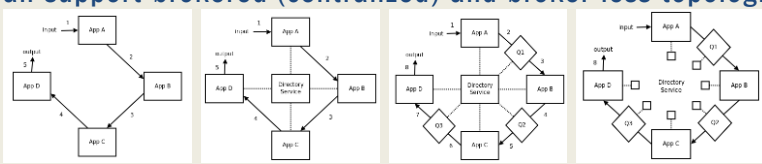
- 4.1 Foundations
  - Protocols
  - Types of communication
- 4.2 Remote procedure call
- 4.3 Message-oriented communication
  - Socket communication
  - Messaging libraries
  - Message-Passing Interface (MPI)
  - Message-queueing systems
  - Examples
- 4.4 Multicast communication
  - Flooding-based multicasting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

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

15

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



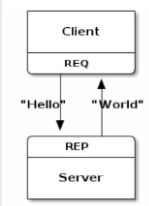
16

ZEROMQ – 2

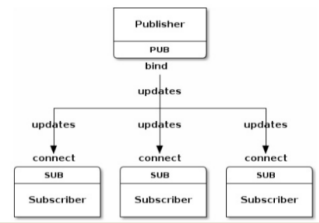
- ZeroMQ is **TCP-connection-oriented communication**
- Provides socket-like primitives with more functionality
  - Basic socket operations ***abstracted*** away
  - Supports many-to-one, one-to-one, and one-to-many connections
  - ***Multicast*** connections (one-to-many – single server socket simultaneously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

ZEROMQ - PATTERNS

- **Request-reply pattern**
  - Traditional client-server communication (e.g. RPC)
  - Client: request socket (**REQ**)
  - Server: reply socket (**REP**)

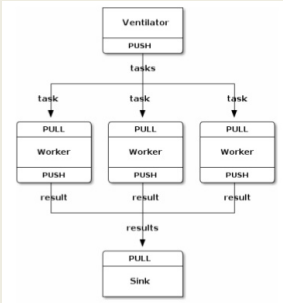


- **Publish-subscribe pattern**
  - Clients **subscribe** to messages **published** by servers
  - As in event-based coordination (Ch. 1)
  - Supports multicasting messages from server to multiple
  - Client: subscribe socket (**SUB**)
  - Server: publish socket (**PUB**)



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



19

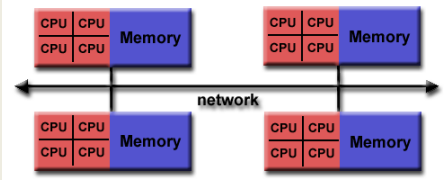
QUEUEING ALTERNATIVES

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

20

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



21

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

22

MOTIVATIONS FOR MPI - 2

- Supercomputers had proprietary communication libraries
  - Offer a wealth of efficient communication operations
- All libraries mutually incompatible
- 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

23

MPI FUNCTIONS / DATATYPES

- Very large library, v1.0 (1994) 128 functions
- Version 3 (2015) 440+
- 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	unsigned 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_GRAPHDIMS_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_IRECV	MPI_IRSEND	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_HVECTOR	MPI_TYPE_INDEXED
MPI_TYPE_LB	MPI_TYPE_SIZE	MPI_TYPE_STRUCT	MPI_TYPE_UB
MPI_TYPE_VECTOR	MPI_UNPACK	MPI_WAIT	MPI_WAITALL
MPI_WAITANY	MPI_WAITSOME	MPI_WTICK	MPI_WTIME

24

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, wat until transmission starts
MPI_sendrecv	Send message, wait for reply
MPI_isend	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!**

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

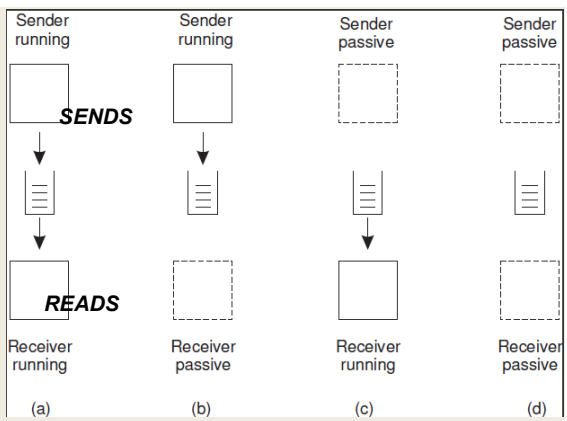
MESSAGE QUEUEING SYSTEMS: USE CASES

- 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

27

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…



28

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 Inteface**
- 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

29

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

30

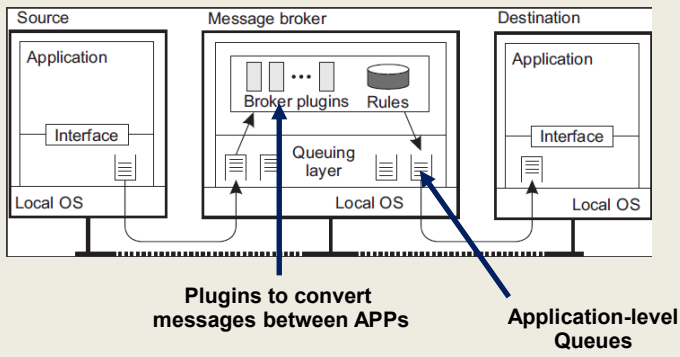
MESSAGE QUEUEING SYSTEMS ARCHITECTURE - 2

- **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

31

MESSAGE BROKER ORGANIZATION



32

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+

33

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

34

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

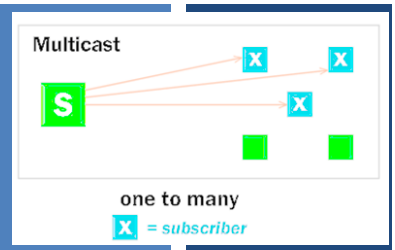
OBJECTIVES – 2/20

- Questions from 2/15
- Assignment 2: Key Value Store
- Assignment 3: Replicated Key Value Store
- Chapter 4: Communication
  - Chapter 4.1: Foundations
  - Chapter 4.2: RPC (light-overview)
  - Chapter 4.3: Message Oriented Communication
  - **Chapter 4.4: Multicast Communication**

37

WE WILL RETURN AT 4:55 PM

38



CH. 4.4: MULTICAST COMMUNICATION

Apache ActiveMQ

L13.39

39

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 20, 2024	TCSS558: Applied Distributed Computing [Winter 2024] School of Engineering and Technology, University of Washington - Tacoma	L13.40

40

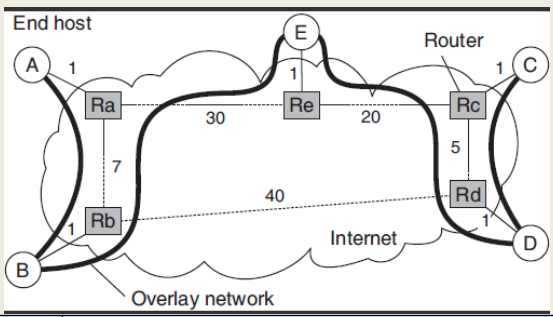
MULTICAST COMMUNICATION

- 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

41

NETWORK STRUCTURE

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



42

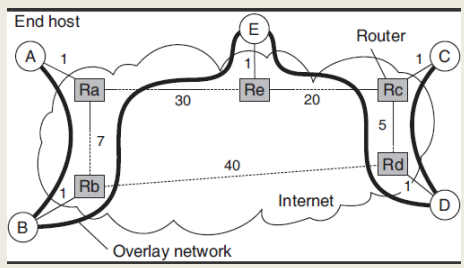
APPLICATION LEVEL TREE-BASED MULTICASTING

- 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

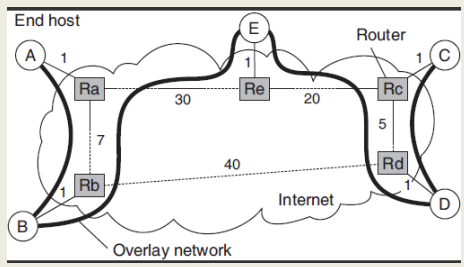
- **Stretch (Relative Delay Penalty RDP)**
- CONSIDER routing from B to C
- ***What is the Stretch?***
- Stretch (delay ratio) = Overlay-delay / Underlying-delay
- *Overlay:* B→Rb→Ra→Re→E→Re→Rc→Rd→D→Rd→Rc→ C = 73
- *Underlying:* B→Rb→Rd→Rc→C = 47
- Stretch = 73 / 47 = 1.55
- Captures additional time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

45

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?

46

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

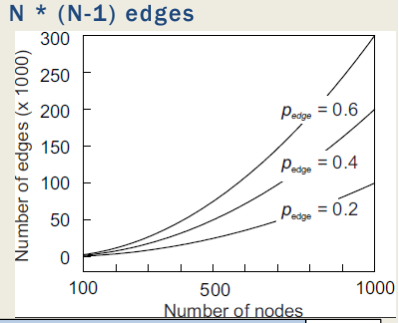
47

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…



48

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

February 20, 2024	TCSS558: Applied Distributed Computing [Winter 2024] School of Engineering and Technology, University of Washington - Tacoma	L13.49

49

PROBABILISTIC FLOODING

- ***….Washington state in winter?***
- When a node is flooding a message, concept is to enforce a prob
- Throtti
- Impler s 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?

February 20, 2024	TCSS558: Applied Distributed Computing [Winter 2024] School of Engineering and Technology, University of Washington - Tacoma	L13.50

50

PROBABILISTIC FLOODING

- ***….Washington state in winter?***
- When a node is flooding a message, concept is to enforce a prob
- Throtti
- 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 = ½ * P_{edge} * N * (N-1)

February 20, 2024	TCSS558: Applied Distributed Computing [Winter 2024] School of Engineering and Technology, University of Washington - Tacoma	L13.51

51

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 = ½ * P_{edge} * N * (N-1)
½ * (.1) * (10000) * (9999)

February 20, 2024	TCSS558: Applied Distributed Computing [Winter 2024] School of Engineering and Technology, University of Washington - Tacoma	L13.52

52

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 = ½ * P_{edge} * N * (N-1)
½ * (.1) * (10000) * (9999)
4,999,500 edges

53

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?

54

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 $p=(1-p_{flood})^n$

55

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?

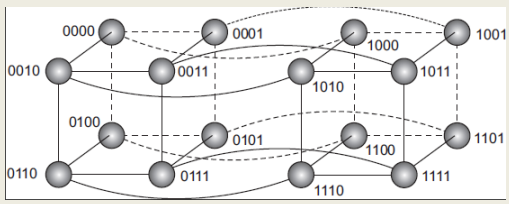
If a node Q has n neighbors, the probability that all neighbors don't forward the message to Q is $p=(1-p_{flood})^n$

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)

56

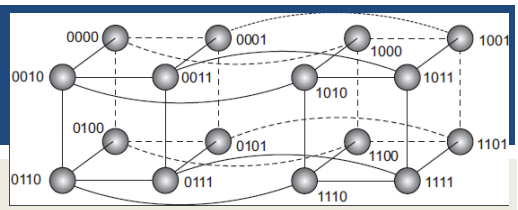
MESSAGE FLOODING

- 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



57

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)

58

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

59

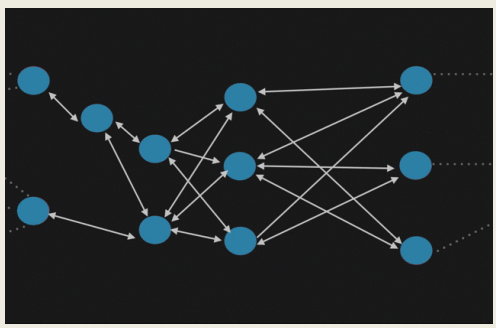
GOSSIP BASED DATA DISSEMINATION

- 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



60

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

61

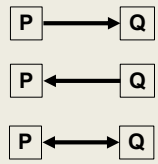
EPIDEMIC PROTOCOLS

- **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

62

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)



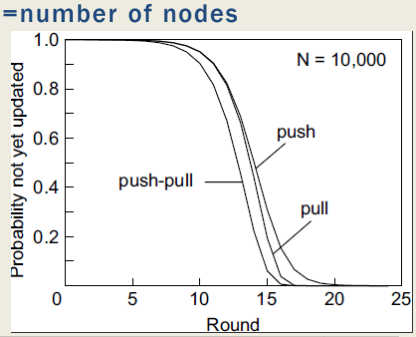
- Push only: hard to propagate updates to last few hidden susceptible nodes
- Pull: better because susceptible nodes can pull updates from infected nodes
- Push-pull is better still

63

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 ith round.
- For pull, push, and push-pull based approaches:

10,000 nodes →



64

RUMOR SPREADING

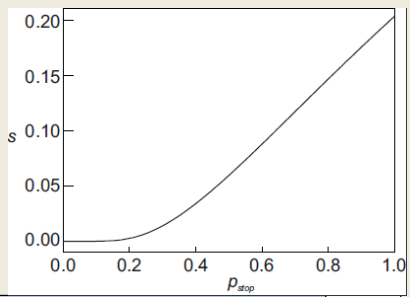
- Variant of epidemic protocols
- Provides an approach to **"stop"** message spreading
- Mimics "gossiping" in real life

- **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)

65

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 →



66

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

67

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**

68

QUESTIONS

February 20, 2024

TCSS558: Applied Distributed Computing [Winter 2024]
School of Engineering and Technology, University of Washington - Tacoma

L13.69

69