Multiparty Communications
CS 118
Computer Network Fundamentals
Peter Reiher

Outline

- Extending 2-party model to N-party
- A party has multiple receivers (other end)
- A party has multiple senders (local end)
- Multiples of information

Shannon Channel

- Two preselected parties
  - Homogenous endpoints

- Unidirectional channel
  - Preselected sender, preselected receiver
- ***One predetermined sender, one predetermined receiver***

Shannon 2-party communication

- We began by knowing:
  - Participating endpoints
  - Communication channel
- We didn't know, but fixed:
  - *When* the endpoints share state
    - So we need a handshake
    - Including "when they want to be active" vs. idle
  - Whether something is *lost*
    - So we need timers

Decoupling party from channel

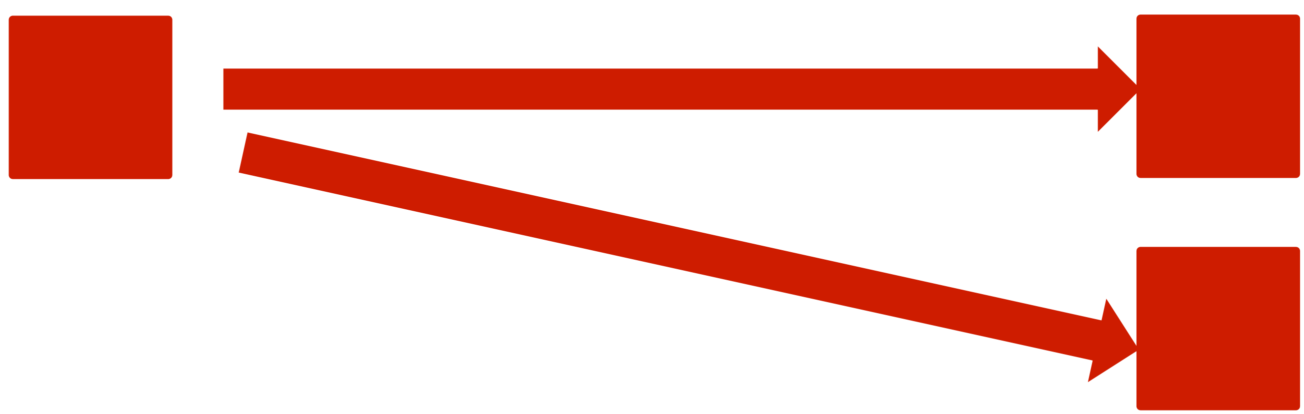
- What if we want to talk to different parties?
  - Sometimes we communicate with Twitter
  - Sometimes we communicate with eBay
  - Sometimes we communicate with Wikipedia
- Don’t want a permanent, always-on channel to each of them
- How can we do better?
  - “Detach” channel end from party

Channel vs. party

- Shannon channel
  - Integrated with the endpoint (party)
  - No choices – all information sent/received uses the only channel there is

Separating the two

- Need to treat what happens in the endpoint (state to share) from the channel (because there might be more than one)



Abstract network components

- Endpoint
  - ("party")
  - Source or sink of state ("information")

- Link
  - ("channel")
  - Action at a distance ("symbol transfer")

Components

Shannon

- Party
- Channel
- Information
- 2-party interaction

Multiparty, modern terms

- Endpoint, node, host
- Link, hop
- State, data
- N-party interaction

Multiparty extensions

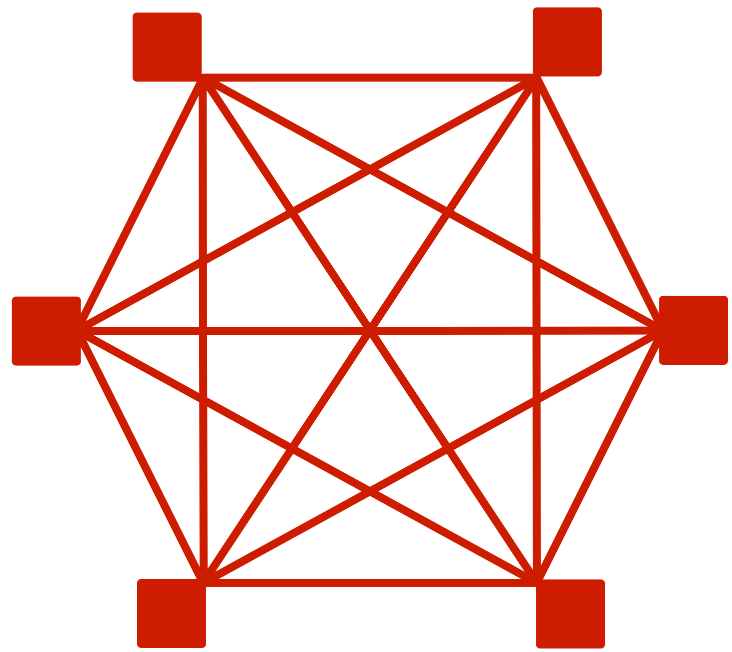
- Which party you're talking to
  - Need to differentiate the receivers
  - Names

- How talk to multiple parties at once
  - Juggling multiple "senders"
  - Sockets

- How to say the same thing multiple times
  - Broadcast and multicast

Multiparty

- Multiple endpoints
  - All connected
  - By separate 2-party channels
  - Using a single protocol

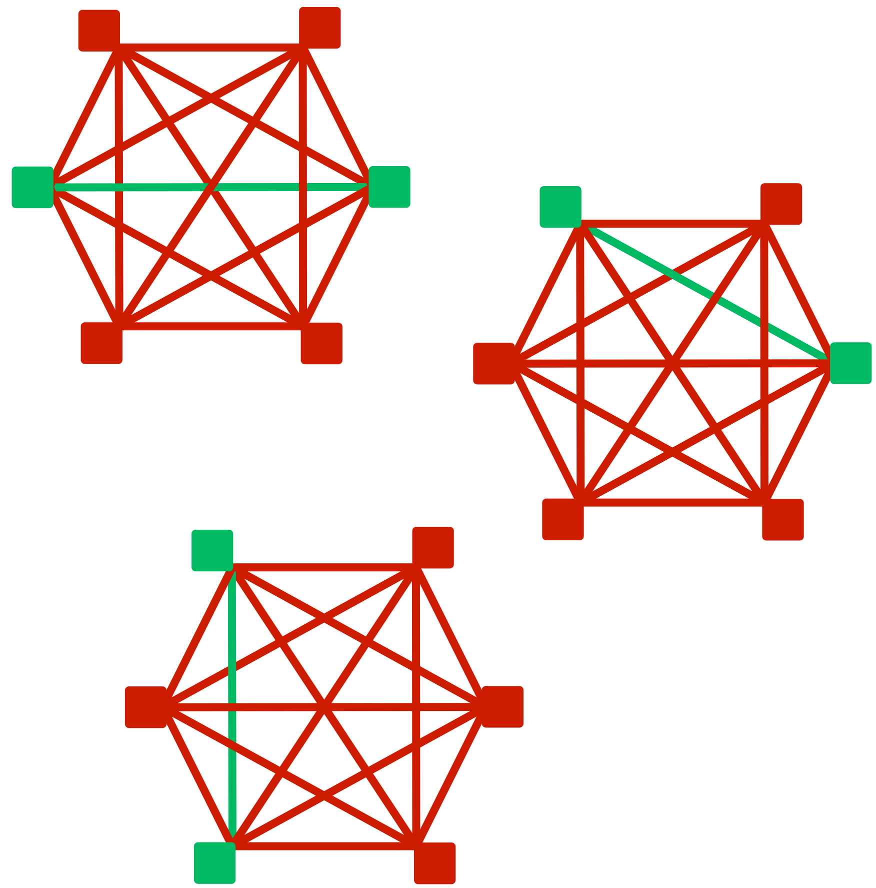


Multiparty assumptions

- Multiple parties
- Using ONE common protocol
- Connected by direct 2-party channels
  – I.e., fully-connected topology
  – Each channel disjoint from the others
    - In state
    - In inputs and outputs

Why is this *networking*?

- Networking
  - Methods to enable communication between ***varying sets*** of indirectly connected parties that don't share a single protocol
- A small increment
  - ONE protocol for now
  - Direct 2-party channels for now
  - (we'll get to the other parts later…)



Importance of multiparty

- Varying participants
  - Pairs communicating change
- Varying view of state
  - Subsets of state, potential overlap, etc.
- More power
  - Can share with more than one other party

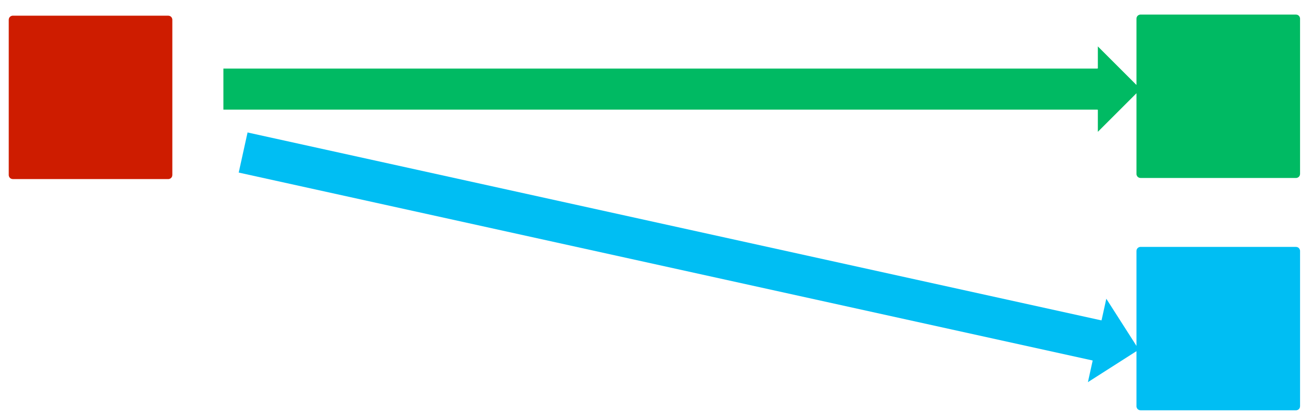
The need for names

- Each source can interact with N-1 receivers
  - How are receivers differentiated?
    - Each uses a different channel
    - But how do we specify which channel is which?

Need some sort of identifier to indicate which channel (indicating which receiver)

A simple case

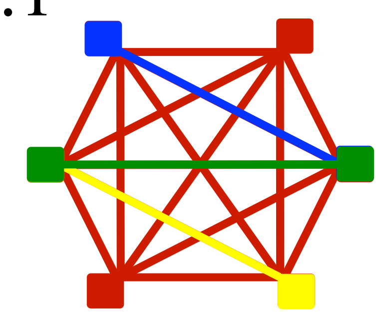
- One sender
- How do we identify one of the two possible receivers?



What can the name apply to?

- Identifier can mean several things at once:
  - Channels
  - Endpoints

- WHY?
  - Consider a fully-connected network
  - For each source, channel:endpoint is 1:1



Names for receivers

- *Index*
  - A number that corresponds to the channel/endpoint
- *Port*
  - An OS-centric type of name specifying what the OS should connect the channel to
- *Channel*
  - Used more generically
- *Socket*
  - Originally (1974 TCP) meant one end of 2-party
  - Unix/BSD copied the term (1983)
  - Now means a LOT more
    - Large data structure with many parts
    - A "socket descriptor", i.e., a pointer to that structure

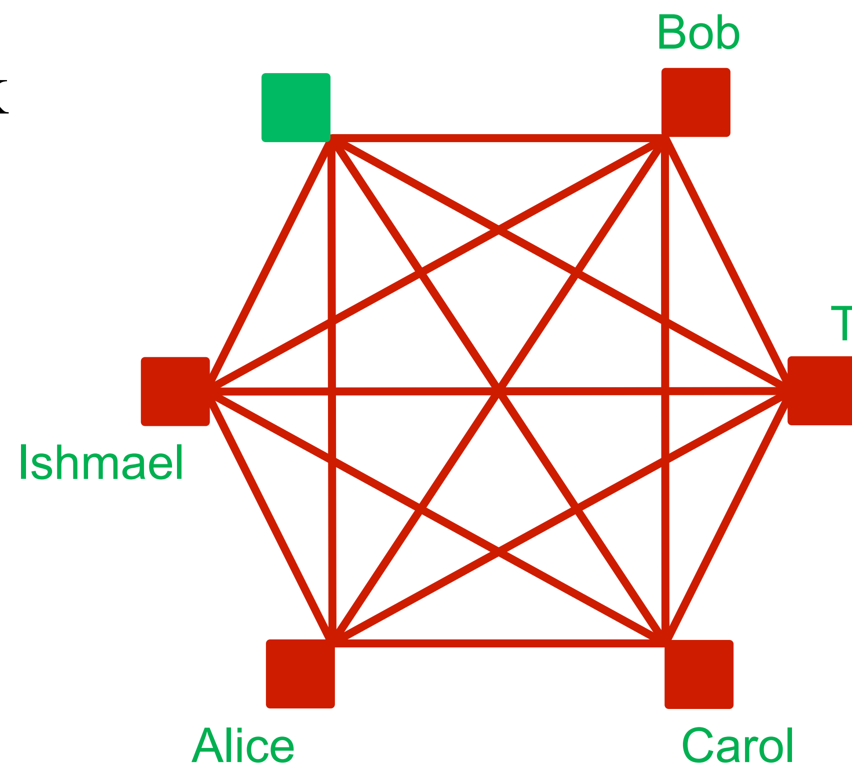
Receiver naming requirements

- How unique?
  - Each party needs to differentiate N-1 receivers
  - Names need to be unique within that set

  - NO need (yet) for names to be unique within the set of all parties
    - You can call me Ray,
      or you can call me J,
      or you can call be Ray J,
      or you can call me RJ, …

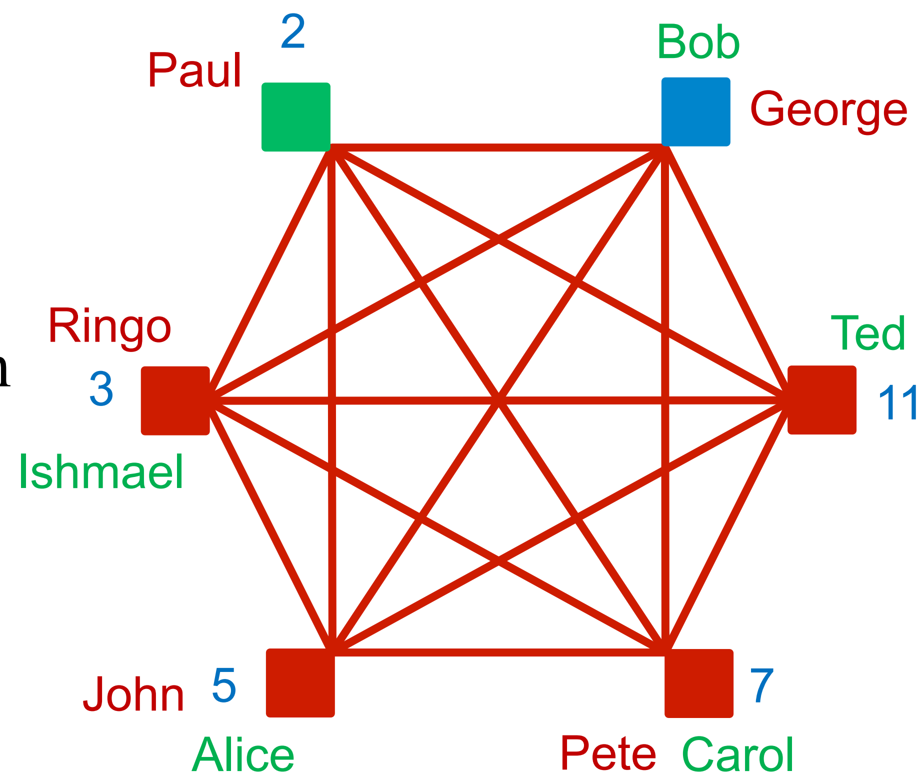
Receiver name examples

- One sender can name the other ends it can talk to



Receiver name examples

- Another sender can do the same thing
  - But possibly with different names
  - Its names need not match anyone else's
- Names are local
  - To the sender and receiver



Multiple senders

- A party can have multiple senders (local end)
- Like my computer talking to multiple web sites

Concurrency

- How does a party deal with multiple communications?
  - The channels – need to "keep 'em separated"
  - Need to decouple the channel from the party itself
- Socket
  - A "disembodied" communication endpoint within a party

What's inside the party?

- Originate/terminate communication
  - State to be shared
- Where's that state?
  - Part of finite state machine (a process) within the party
  - Outside the party
    - We can treat this as output/input of a FSM that relays that info to the channel

How many machines are there?

- Strictly, one
  - Multiple FSMs can be modeled as one FSM
- Simpler to think of them as independent
  - A set of FSMs, running concurrently
    - Multiprocessing
  - And/or running ***as if*** concurrent with each other
    - Multiprogramming
  - And/or having ***internal*** concurrent components
    - Multitasking / multithreading

So what else do we have to name?

- On the machine (or state)
  - Process/thread identifier
  - State identifiers
- Why?
  - Need to know which portion of the party’s state interacts with a given channel

Internal naming requirements

- How unique?
  - Each party needs to differentiate some number of "FSMs" (sets of states)
  - Names need to be unique within that set
  - NO need for names to be unique within the set of all parties
    - Will there ever be such a need?
    - State is always local to the endpoint

Summary of multiparty naming

- Need a way to pick an outgoing channel/receiver
  - An internal channel index
- A way to pick a subset of internal state/machine
  - An internal machine index

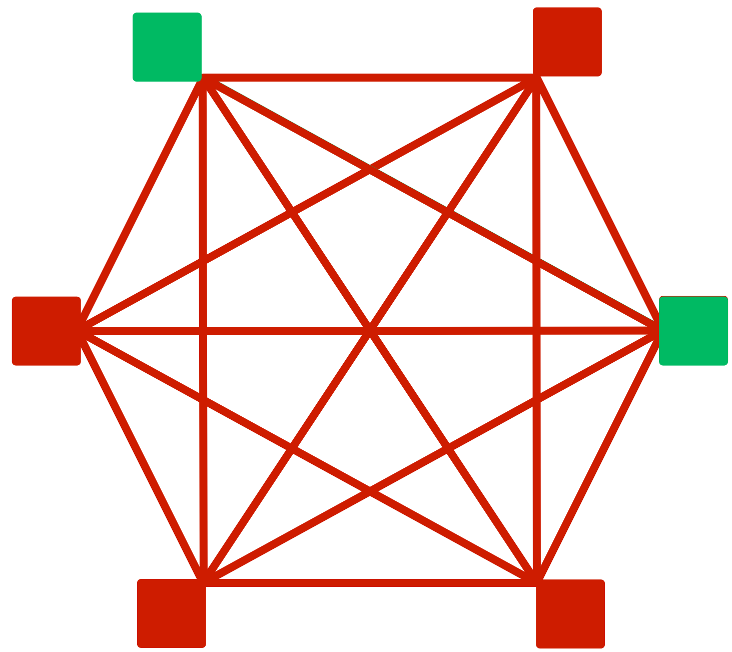
BOTH ARE INTERNAL ONLY

Multiples of communications

- Each party usually wants to communicate to multiple other parties
- Sometimes 1-to-1
- Sometimes same info to many others

Shannon channel

- Unicast
  - 1:1
- Two parties share state
  - Pick which two
  - Just communicate
- State now shared!

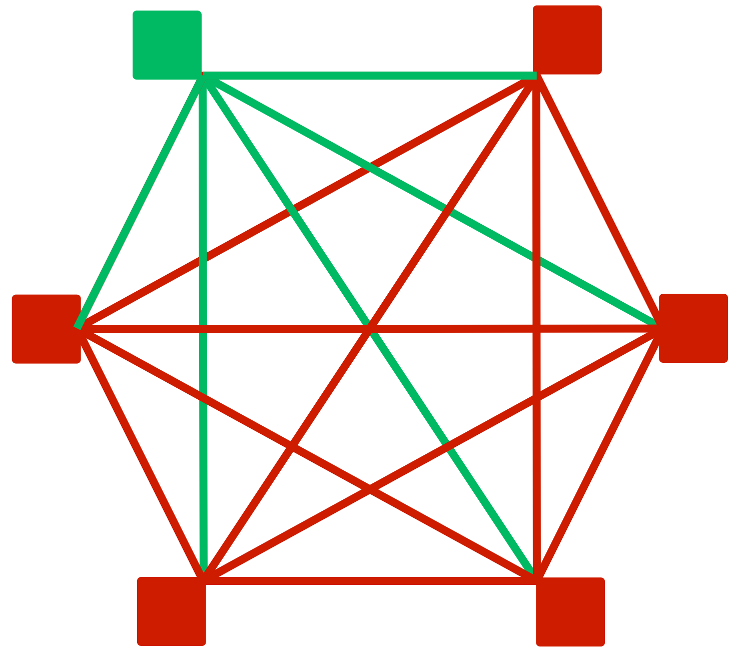


Multiple receivers

- Broadcast (1:N)
  - Send same info. on all channels
  - Every party in the network has the same info.
- Multicast (1:M)
  - Broadcast on a subset of channels

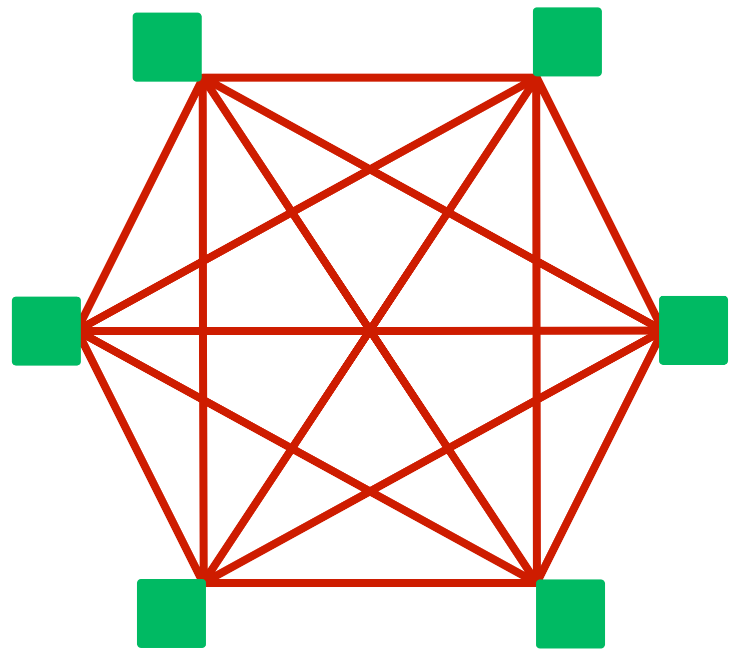
Broadcast

- Share state everywhere
  - No need to pick
  - Need to replicate
    - Multiple communication
    - Multiple information



Broadcast

- State now shared
  - When?
    Need to coordinate
  - How to coordinate?
    - Three-way handshake
    - Chang's "Echo alg."

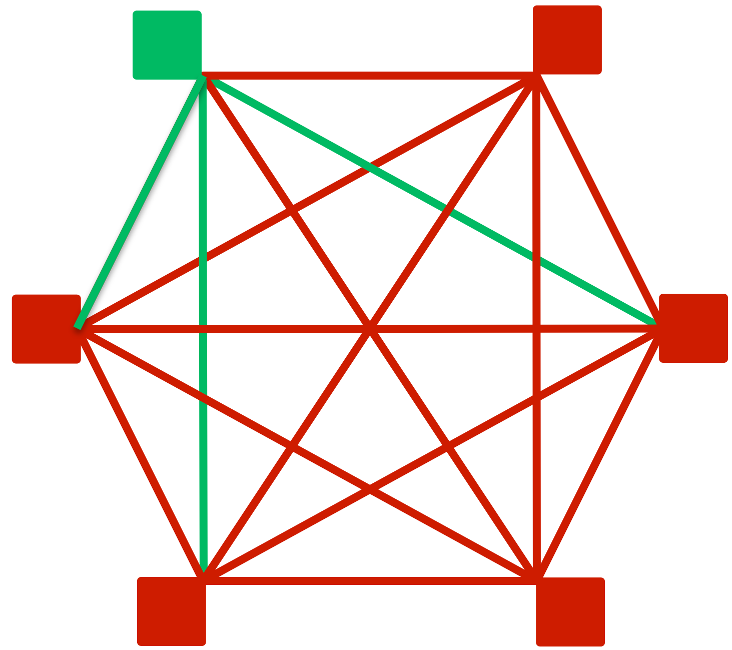


Complexities of communications copying

- Atomicity
  - Losses don't correlate across channels
  - Might link "all-or-none" behavior
- Synchrony
  - Knowing all the receivers have the info at the same time
  - Having them know that
  - Having you know that
- Efficiency
  - Send one to each receiver? Can we do better?

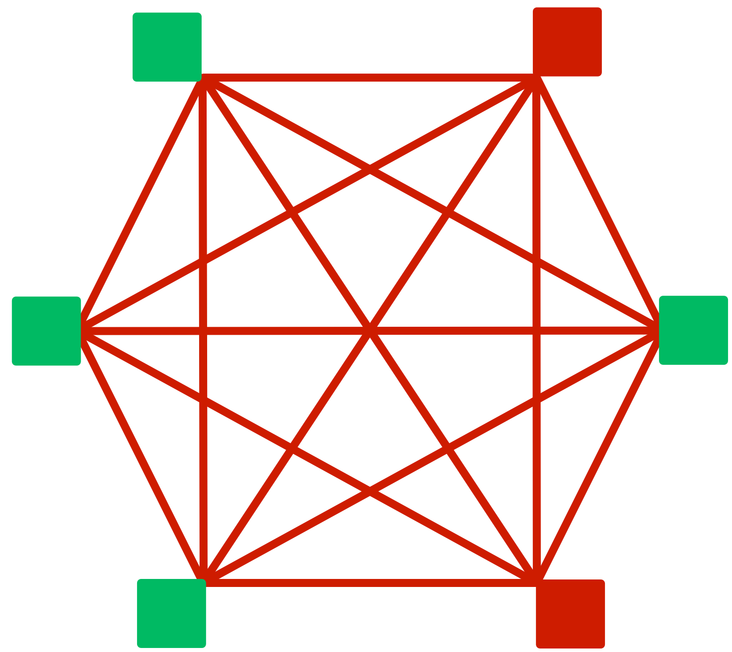
Multicast

- Share with a subset
  – How to pick?
  – Who picks?
- Similar to broadcast
  – Need to replicate
  – Need to coordinate



Multicast

- Things get worse…
  - Subset can change
    - Add parties
    - Remove parties



Multicast complexities

- Group selection
  - How do you indicate the subset desired?
  - Who picks? Sender or receivers?
- Changes in group
  - Members join
  - Members leave

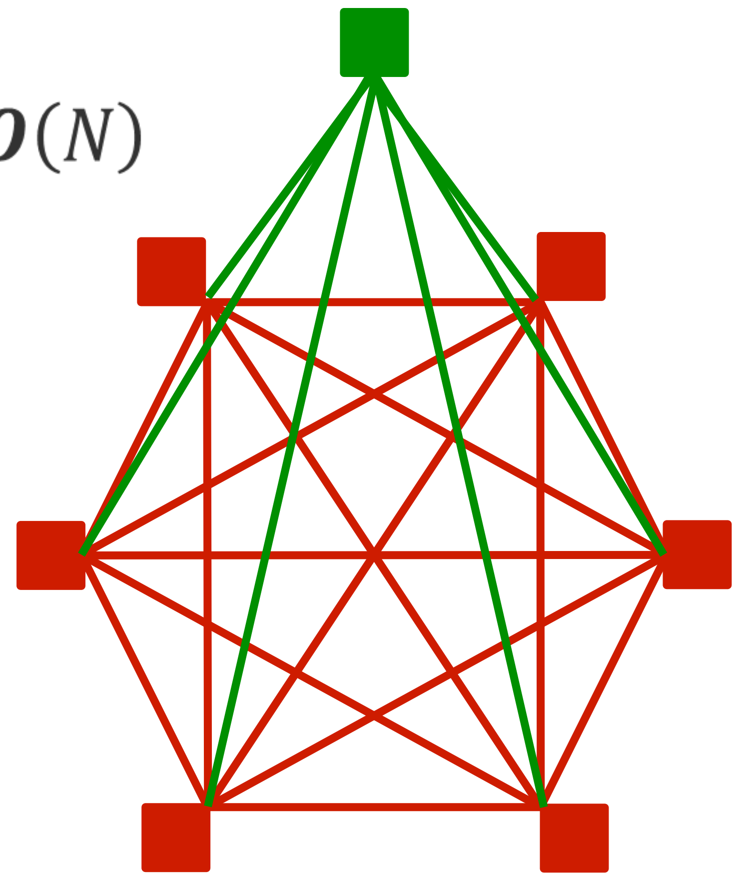
Full pairwise connectivity

- One topology
  - Full, 1-hop connectivity
  - Simple to understand
- Expensive to maintain and use
- Hard to add new members

Problems with this picture

- Fully connected
  - Cost to add one node = $\boldsymbol{O}(N)$
  - Total cost = $\boldsymbol{O}(N^2)$

Solution: sharing



What can we share?

- Endpoints
  - We’re already doing that
  - Multiprocessing, multiprogramming, etc.
  - The rest is for CS 111 (Operating Systems)
    - Virtualization (abstraction!)
    - Resource sharing within a FSM
- Channels
  - Let’s explore…

Sharing a channel

- Sharing in different directions
  - Full-duplex
- Shared outgoing destination
  - A way to support broadcast/multicast
- Shared incoming source
  - To gather information from multiple sources

The big reason

Scale

Scale

- A relationship between two variables and their ratio
  - An independent variable that changes arbitrarily
  - A dependent variable that is expressed in terms of the independent one

    $y = f(x)$

- The ratio grows in some way: $\frac{y}{x} = \frac{f(x)}{x}$

  $\frac{f(x)}{x} \leq c * g(x)$ where c is a positive constant

- We say $f(x)$ is bounded by $O(g(x))$

Scale magnitude

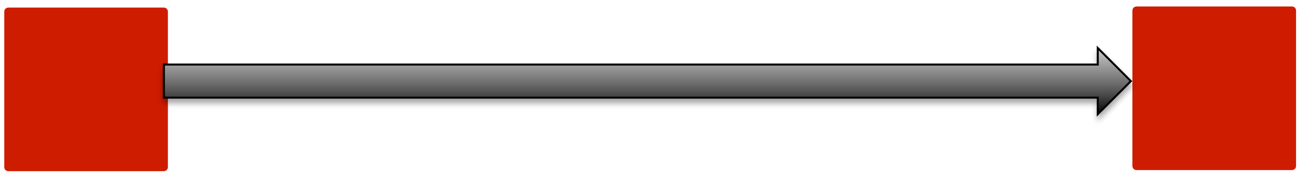
- Growth is bounded
  - No increase
    - Unlimited messaging at no extra cost
  - Logarithmic increase $y = c \log_k x$
    - Phone numbers –one digit gets 10x more numbers
  - Linear increase $y = cx$
    - 6 phones cost roughly 6x one phone
  - Polynomial increase $y = cx^k$
    - Every new person in the room adds N possible pairings
  - Exponential increase $y = ckc^x$
    - Not as bounded!
  - Beyond exponential increase $y = cx^{cx}$
    - Even worse, like factorial

Why do we care?

- Metcalfe's law
  - Value of a network is related to the number of pairwise opportunities
  - I.e., for N nodes, value is N^2
  - In a fully-connected network, cost is N^2
  - Ratio of value to cost is 1:1
    - Can we do better?
    - Can we make the network cost grow more slowly than the increase in value?

2-party sharing

- 2-party channel

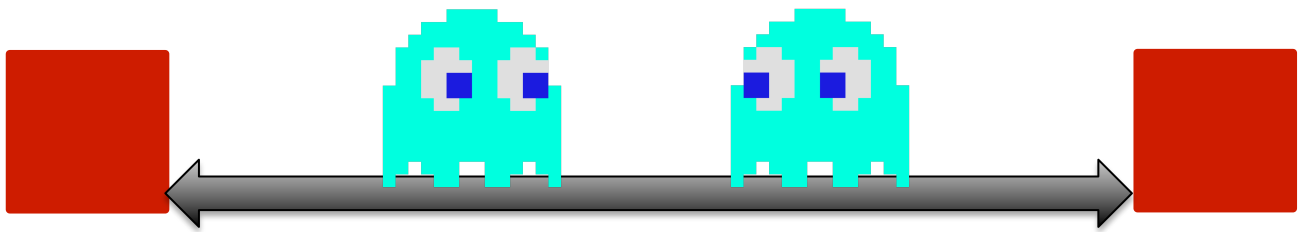


- Let's make it two way:

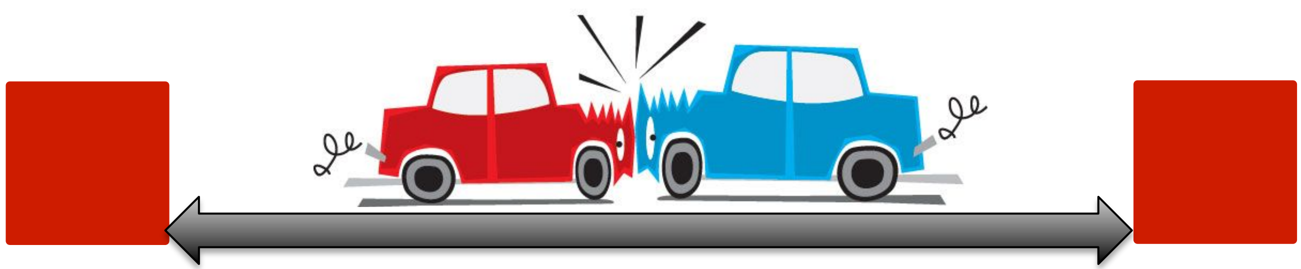
- How?

Signals in different directions

- Some types of particles don't interfere
  - Bosons: pass right through each other

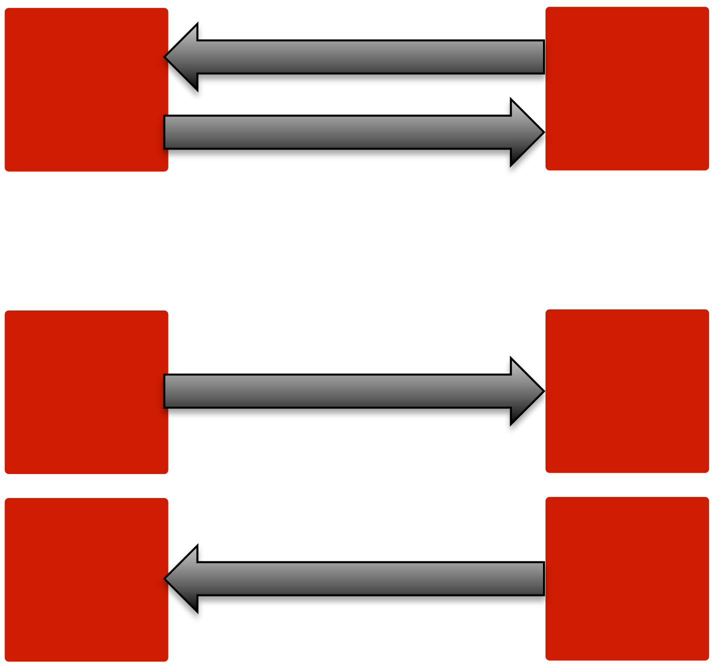


- Others do interfere
  - Fermions: collide (Pauli exclusion principle)



For those that interfere,

- Keep them separated
- By space
  - Two simplex channels
  - Back where we started!



- By time
  - “Timesharing”
  - Time-division

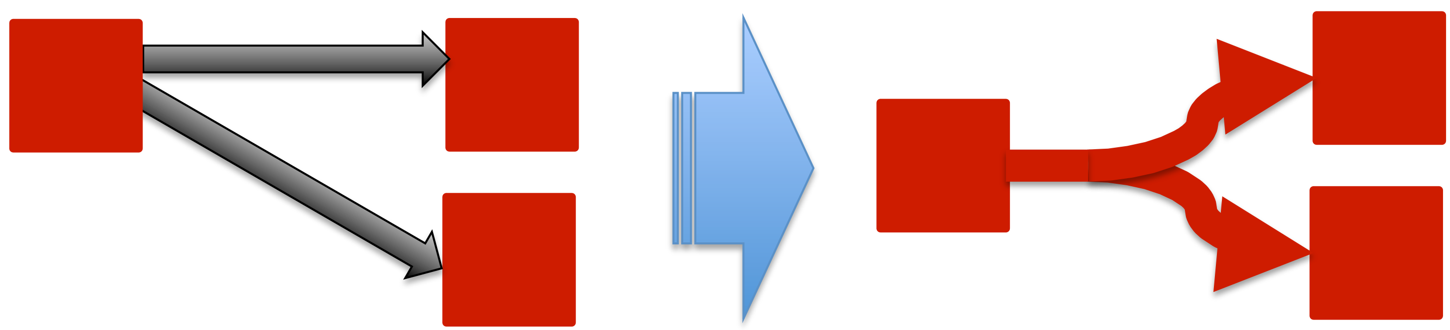
Time sharing control

- Prior agreement
  - I.e., embedded in the protocol description
  - Requires a common time event (synchronization)
- Central controller
  - One side controls the communication

We'll see more general cases later

N-party sharing: 1 to N

- Share an outgoing channel
- One channel to several destinations



1:N – How?

- Receivers *all* see what transmitter sent
  - "Non-destructive" reads
- Which receivers accept the symbols?
  - *All of them* ("native" multicast/broadcast)

Non-destructive reads

- Read by one receiver doesn't affect others
  - Typical case
- Two ways:
  - Groups of identical symbols (e.g., particles)
  - Perfect copies (measurement doesn't alter value)
- Allows sharing to assume broadcast messages
  - Can simplify the sharing protocol

Destructive reads

- Read by one (or a subset) of receivers
  - Rare
- How?
  - Observer effect (read affects value)
  - E.g., quantum state, collect majority of particles, etc.
- Usually considered undesirable
  - Non-determinism – can't control which receiver reads
  - Prevents using broadcast for sharing protocol
- Can be useful for security
  - Tamper evidence if expect only one receiver
  - Quantum cryptography, e.g.

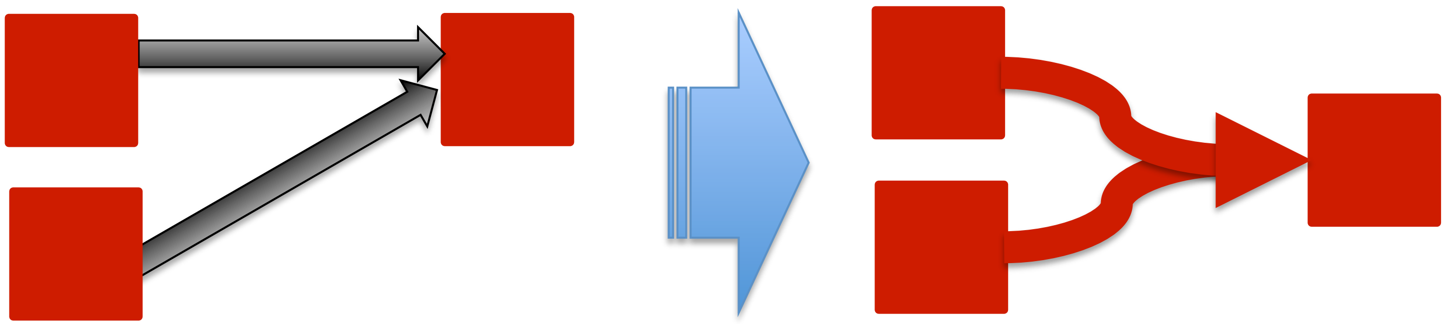
Limiting 1:N transmissions

- How can a sender control which receiver gets the message?
  - Transmit on different channels
  - Transmit at different times
  - Transmit different symbol sets (“languages”)
  - Label the transmission <u>*destination*</u> (names)

All can be internal to the source
I.e., this is the easy part

N-party sharing: N to 1

- Share an incoming channel
- One channel from several sources



N:1 – How?

- Receiver sees what *all* transmitters sent
  - Technically difficult, at the particle level
  - Collisions between particles
  - Or confusion of who sent which particle
  - *One* of them
    - But which one?

Limiting N:1 transmissions

- How can transmitters avoid collisions?
  - Transmit on different channels
  - Transmit at different times
  - Transmit different symbol sets ("languages")
- How can a receiver determine transmitter?
  - (all of the above)
  - Label the transmission *source* (names)

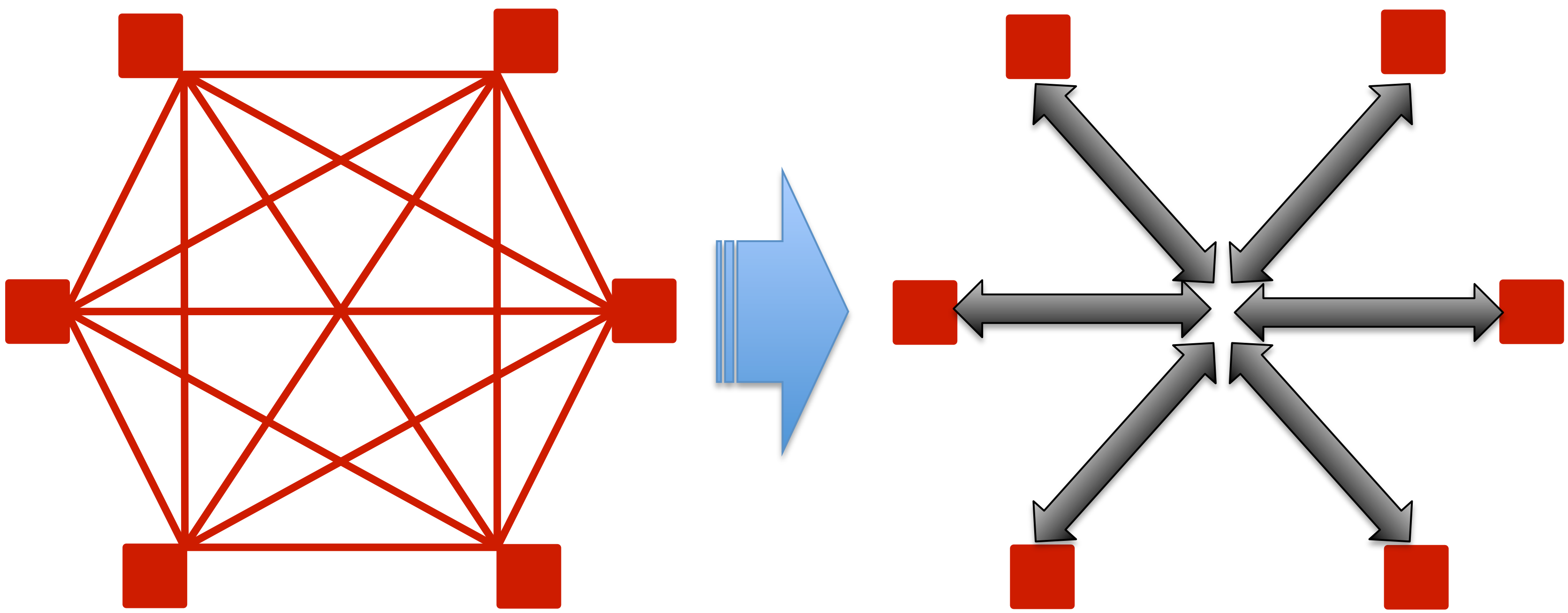
Why is this harder than 1:N?

N:1 is harder than 1:N

- 1:N
  - Coordinate use internal to the source
    - Time, symbol set
  - Naming needs to be coordinated with receiver
    - Need to use IDs the receiver recognizes
    - But each set is unique in the context of that sender
- N:1
  - Coordinate use *between* sources
    - Time, symbol set
  - Coordinate naming
    - Converse of 1:N naming, but name attached by sender
    - How does sender know it has a unique name?

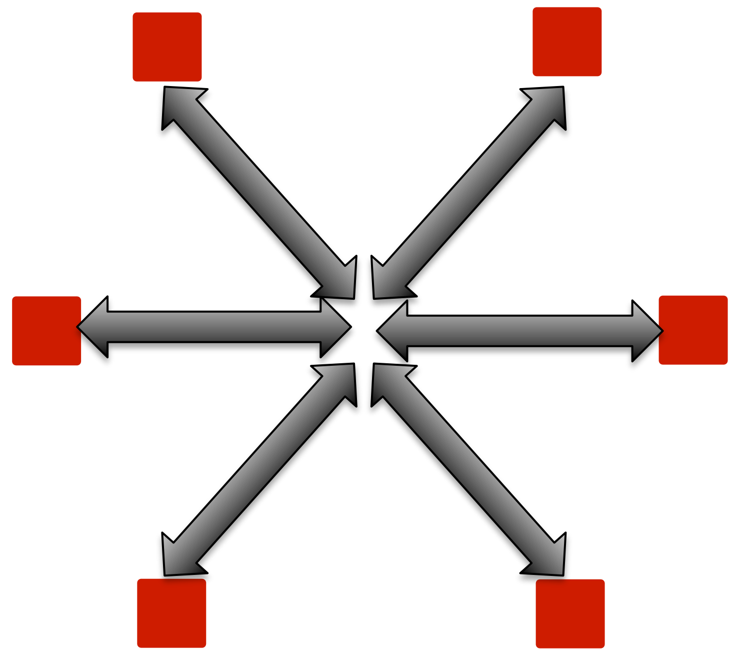
N-party sharing: N to N

- Instead of N^2 links
- Use just N



The ultimate shared channel

- One channel
  - All parties transmit on
  - All parties receive from
- Minimizes link cost
  - One link to add one node



Single shared channel examples

- Freespace
  - Diffuse infrared
  - Omidirectional RF
- Wired
  - Bus
  - Ethernet
- Fiber
  - Individual fibers to a passive coupler

N:N – combine rules

- 1:N – control receiver
  - Transmit on different channels
  - Transmit at different times
  - Transmit different symbol sets (“languages”)
  - Label the *destination*
- N:1 – avoid collision (control transmitter)
  - Transmit on different channels
  - Transmit at different times
  - Transmit different symbol sets (“languages”)
- N:1 – identify source
  - (all of the above)
  - Label the *source*

Summary

- Channel sharing affects network size
  - Distance, number of parties
- Shared channels requires shared namespaces
  - Networking required internal names
  - Sharing requires coordinated names
- Sharing requires mechanism
  - Protocols to manage the network, not just to share endpoint state