Pocket V2N Protocol

An adaption of the Rain Tree Gossip Protocol for the Pocket Network Consensus Module

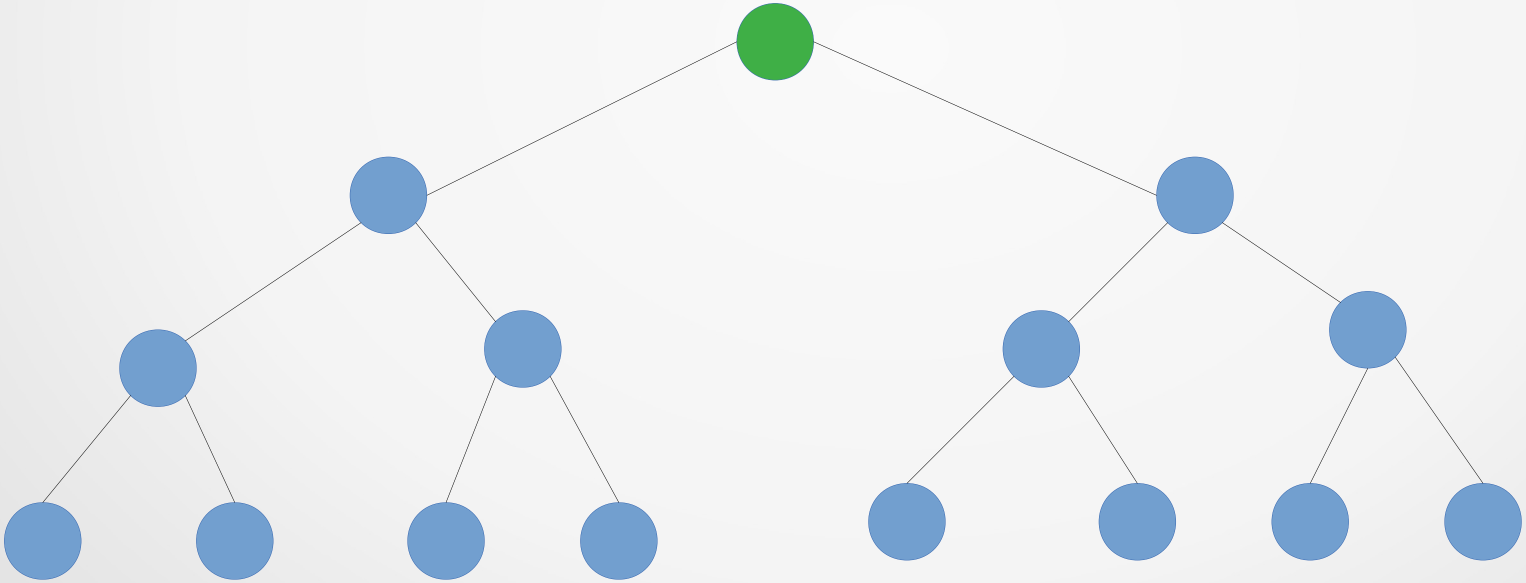
Rain Tree Gossip is:

- Binary Tree distribution with 3 branches*
- ACK/Adjust/Resend at top level to insure against loss of trunk or major branches.
- Optional redundancy layer to insure against non-participation and incomplete lists without the ACK/Resend overhead.
- Double Dasisy Chain clean-up layer to insure 100% message propagation in all cases.

* For clarity of presentation, the discussion and diagrams in the first part of this document all reflect 2 branch trees. Three branch begins at slide 13.

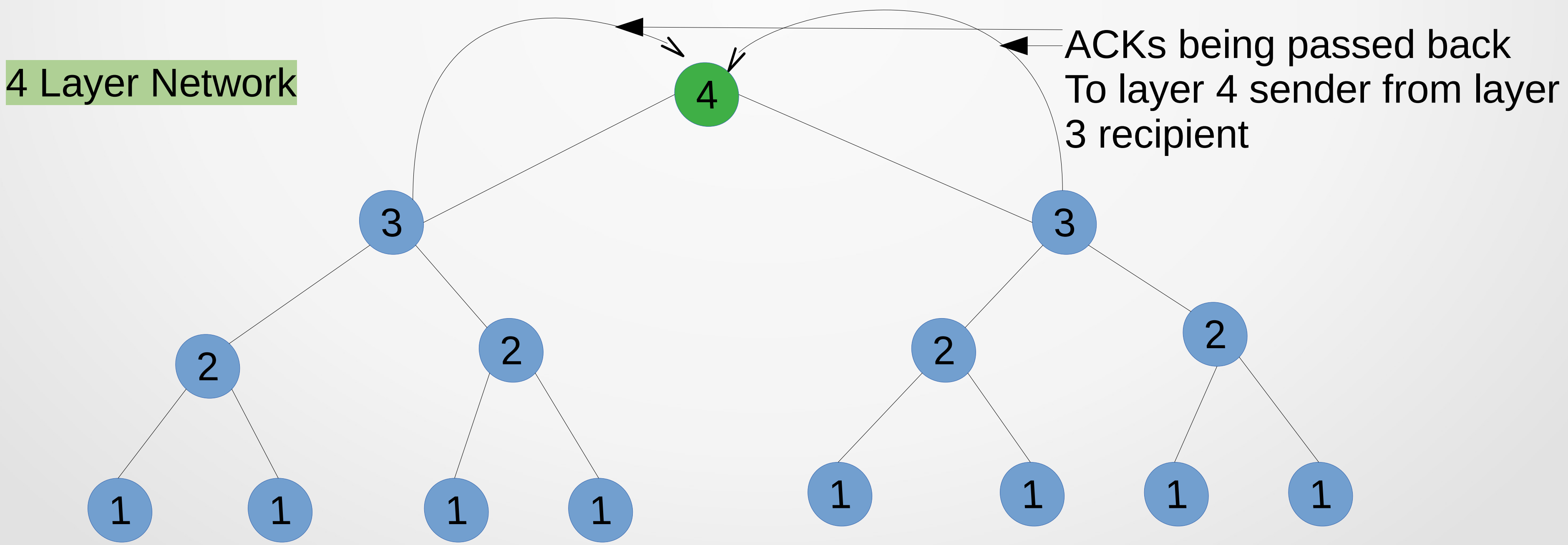
Binary Tree...

- For any set of sorted, randomly distributed data, Binary Tree is the fastest possible lookup method.
- Rain Tree requires that all participant nodes have a sorted list (partial or complete), of target nodes.



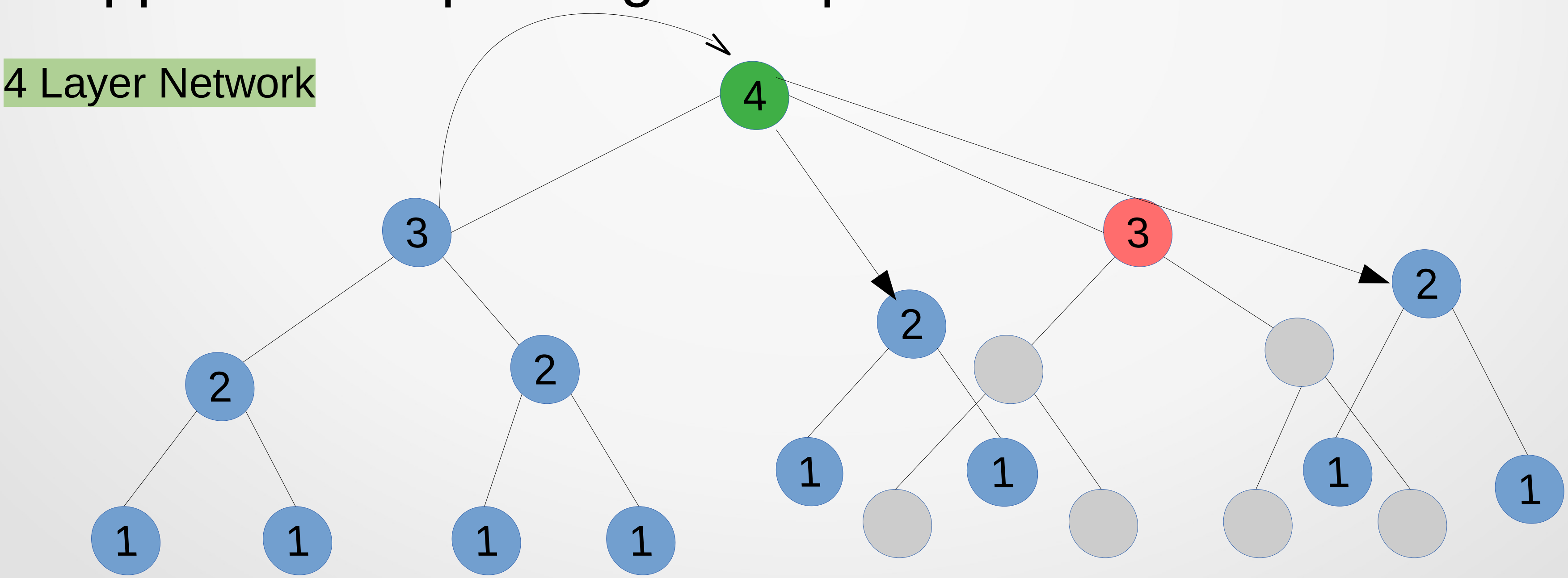
ACK/Adjust/Resend

- Rain Tree messages are sent with logical level numbers.
- Origin level number is determined by the message source node. (= highest position bit in size of list)
- Level number is decremented by one when passed down.



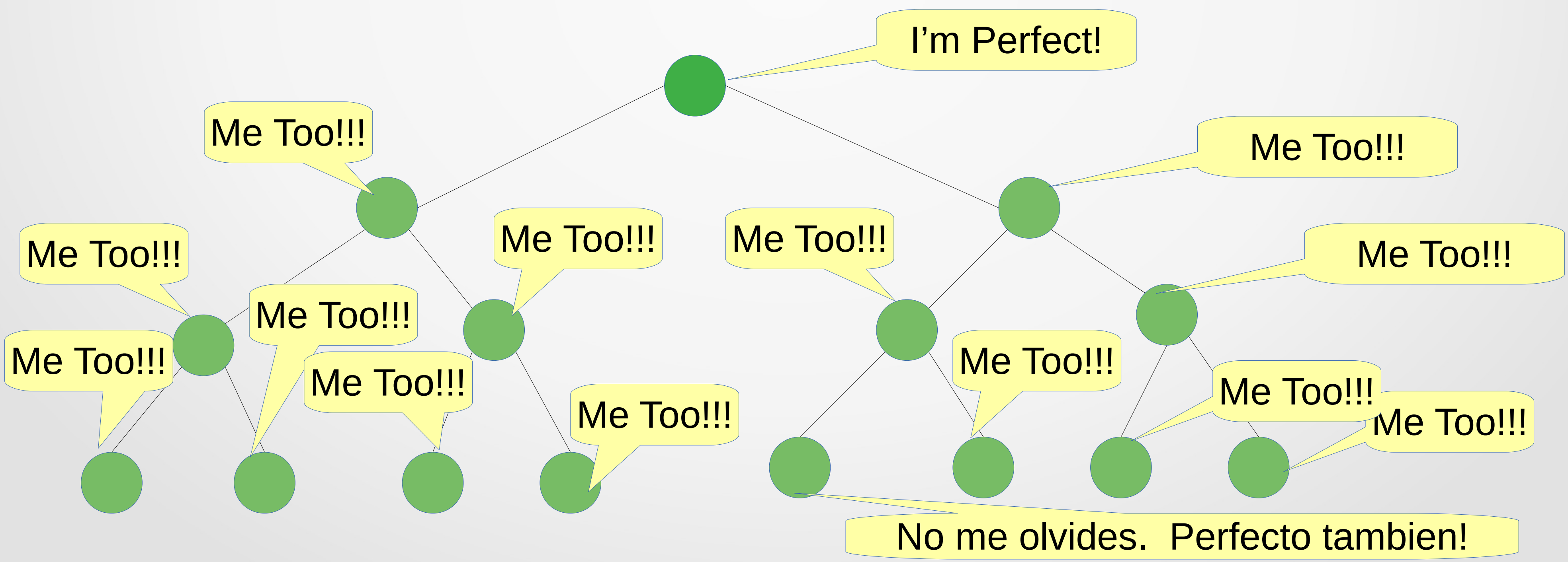
ACK/Adjust/Resend (pg 2)

- Failure to receive an ACK within time-out causes sender to select next 2 nodes (+1 & -1) in list and resend with decremented level number.
- Due to the “tricky” nature of time-outs, Adjust/Resend only happens once per target. Replacement nodes do not ACK.



Department of Redundancy Department

- IF… Every node had the exact same list AND…
- IF… Every node was 100% reliable AND…
- IF… 100% of communications were successful
- THEN… Redundancy would not be desirable.

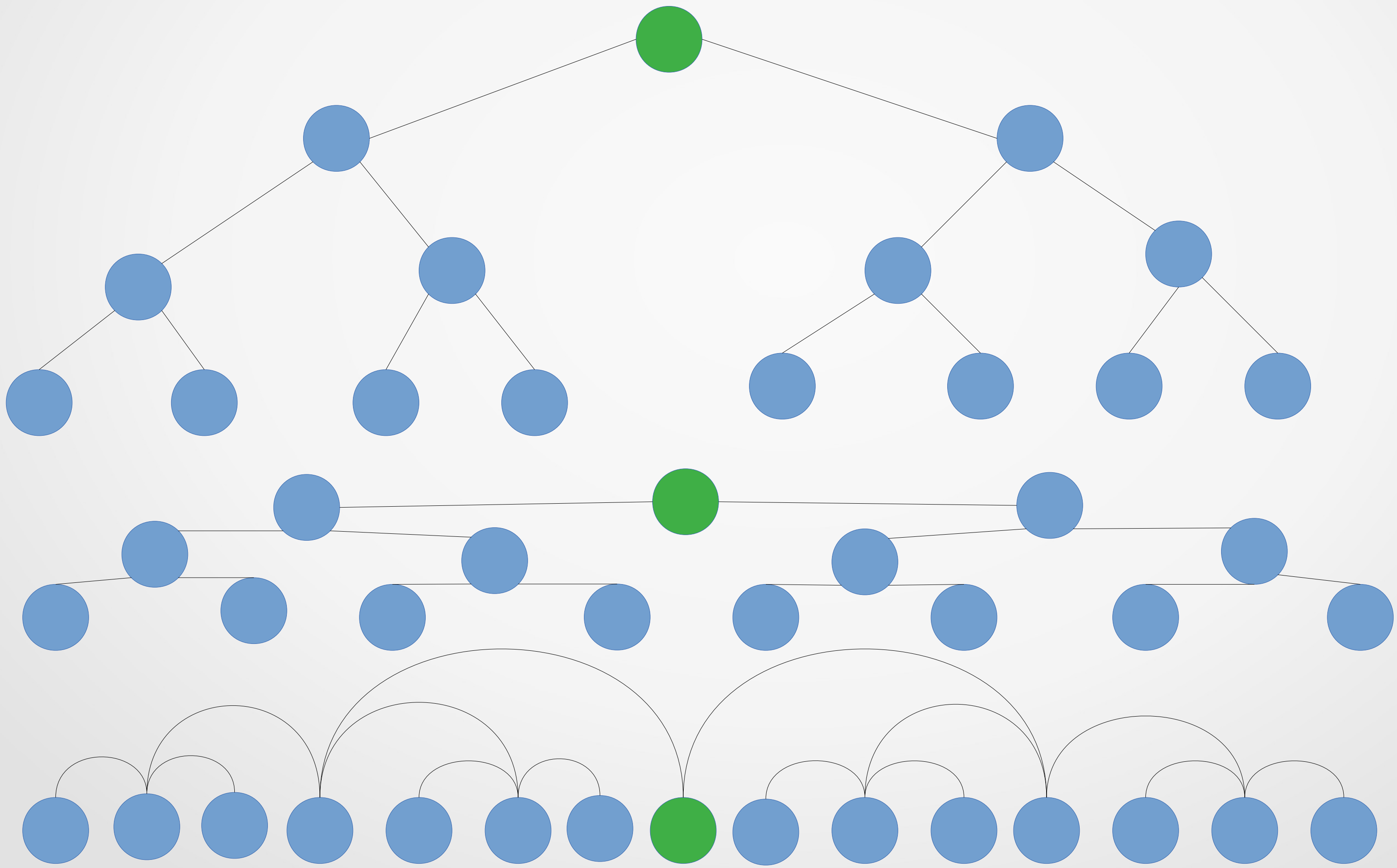


Meanwhile… in the real world..

- Communications fail! Networks don’t actually work the way that simulations often assume they will.

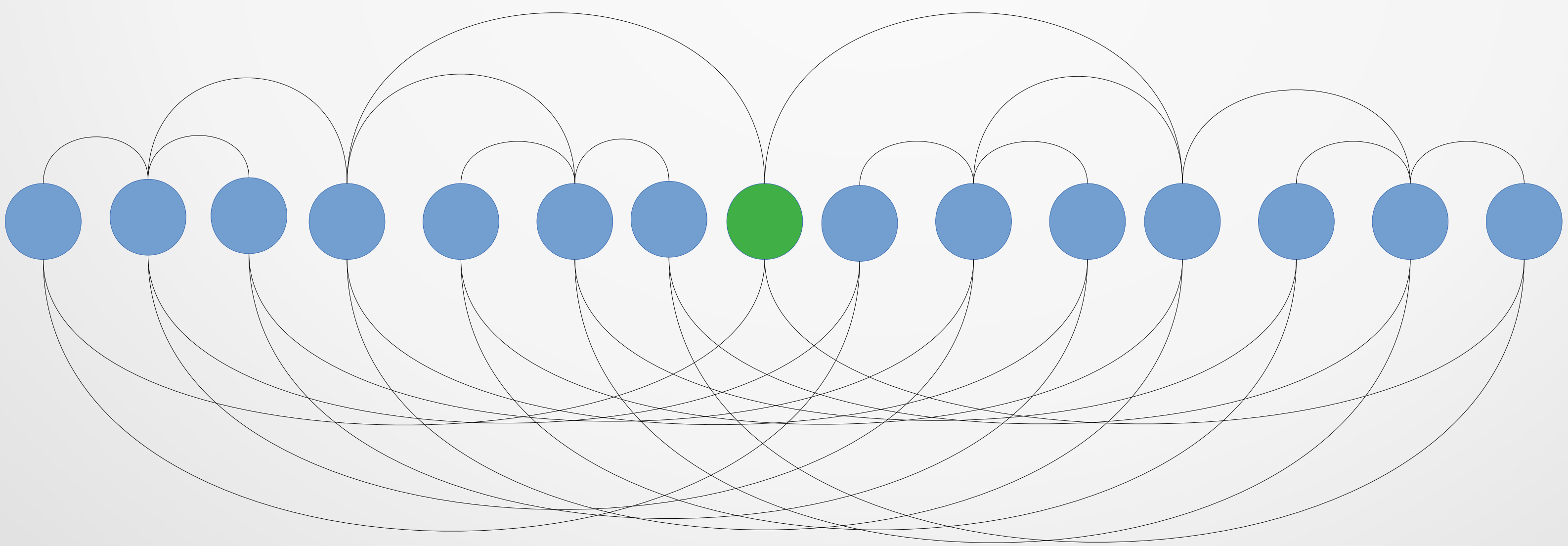
A short pause to reorganize the drawing (smoke 'em if you got 'em)

- We will flatten the drawing to make it look more like a list. Because that's what it really is. Then redraw it to show the same information in a more useful format



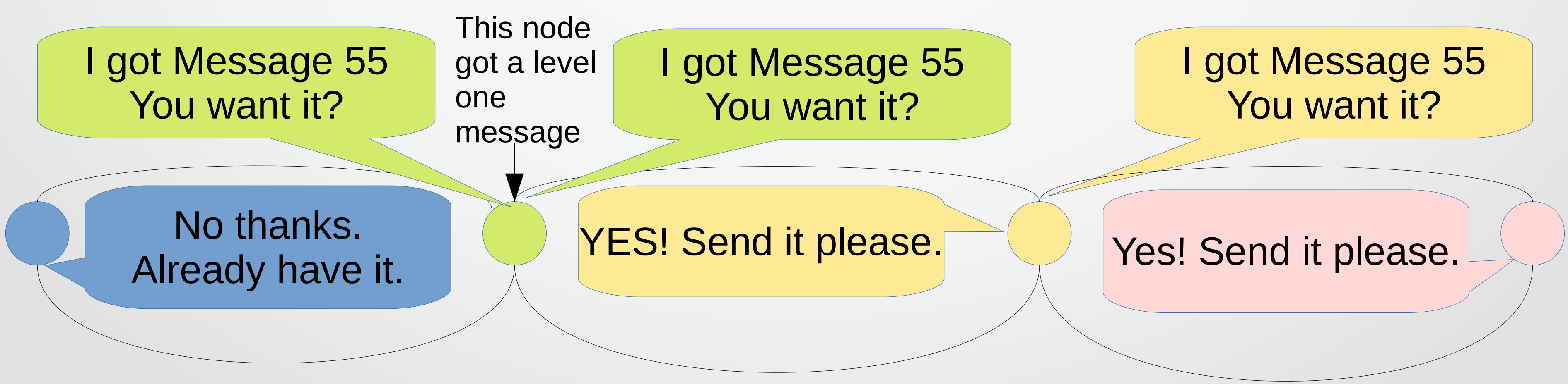
Now then… Let's get redundant.. again

- This sample shows one additional redundancy layer of 1 comm per node.
- Initial distribution required: 14 comms + 2 acks and took 3 ticks to complete.
- Redundancy layer required 15 comms and took 1 tick.



Dasie Chain Clean-up & list maintenance layer.

- When a message reaches layer one, the receiving node sends a layer zero IGYW (I got. You want?) message to its right hand and left hand peers.
- When a node receives an IGYW, it checks the message hash vs. its recently received messages and responds with a yes/no message.
- Latency and/or lack of response are applied to peer and list management if appropriate for the application. If the answer is: “Yes. I want it”, the message is passed on.
- This Bi-Directional Dasiy Chain counts as two additional redundancy layers and insures eventual 100% delivery to all functional nodes.



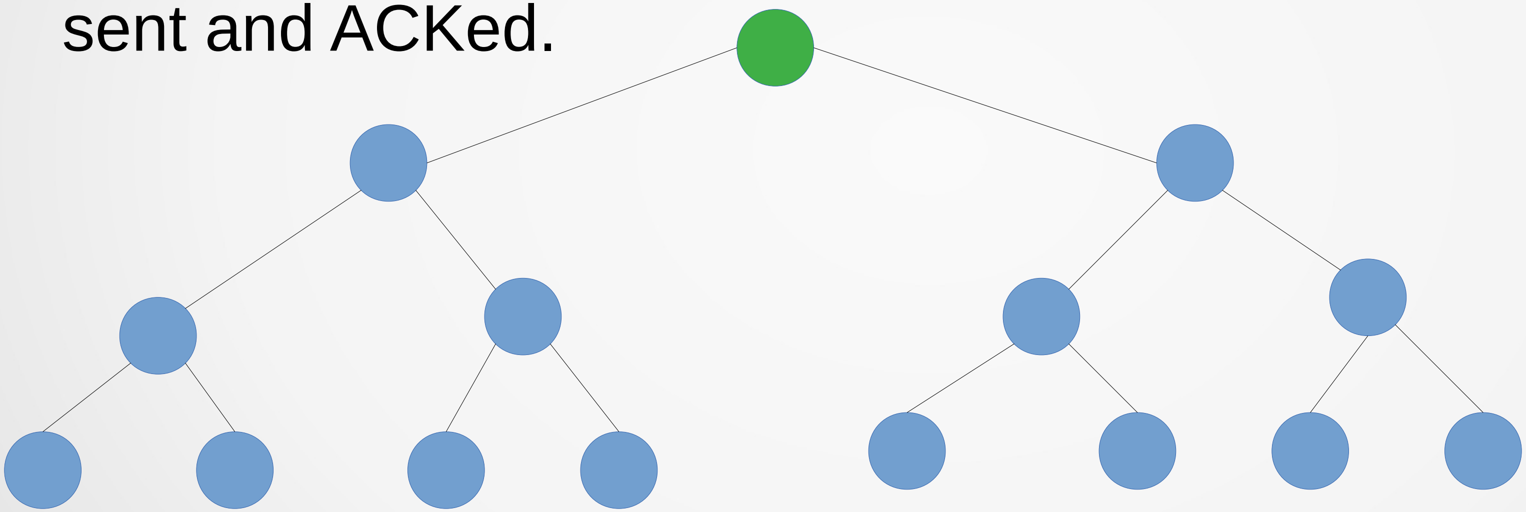
Let’s Review: Rain Tree is...

- Binary Tree distribution with 3 branches*
- ACK/Adjust/Resend at top level to insure against loss of trunk or major branches.
- Optional redundancy layer to insure against non-participation and incomplete lists without the ACK/Resend overhead.
- Dasiy Chain clean-up layer to insure 100% message propogation in all cases.

* For clarity of presentation, the discussion and diagrams in the first part of this document all reflect 2 branch trees. Three branch reasoning is discussed next.

The Third Man. Structural Redundancy durring distribution

- A close look at our binary chart reveals an interesting fact.
- The Green Node (message originator) is divorced from the distribution process as soon as his two messages have been sent and ACKed.

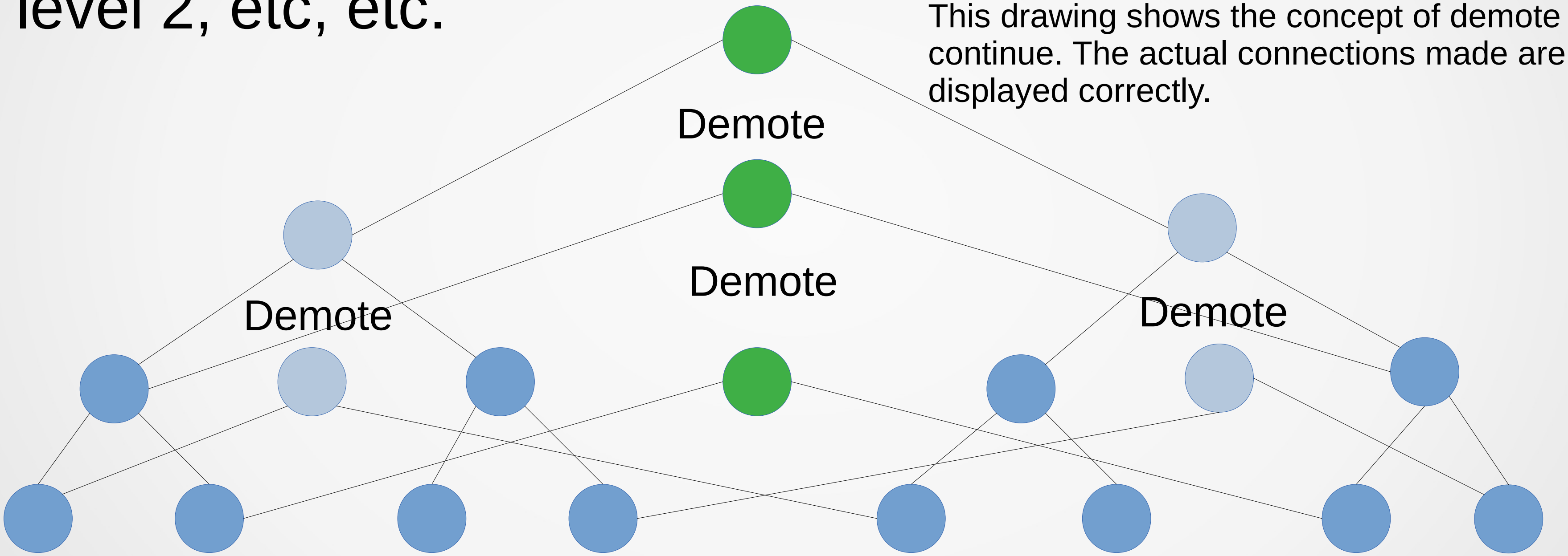


- But, what if we leverage the fact that he is by definition:
- Available, informed, capable and cooperative?

The Third Man. Continued

- After completing his duties at level 4, he demotes himself to level 3 acts as he should, then self-demotes again to level 2, etc, etc.

This drawing shows the concept of demote and continue. The actual connections made are not displayed correctly.



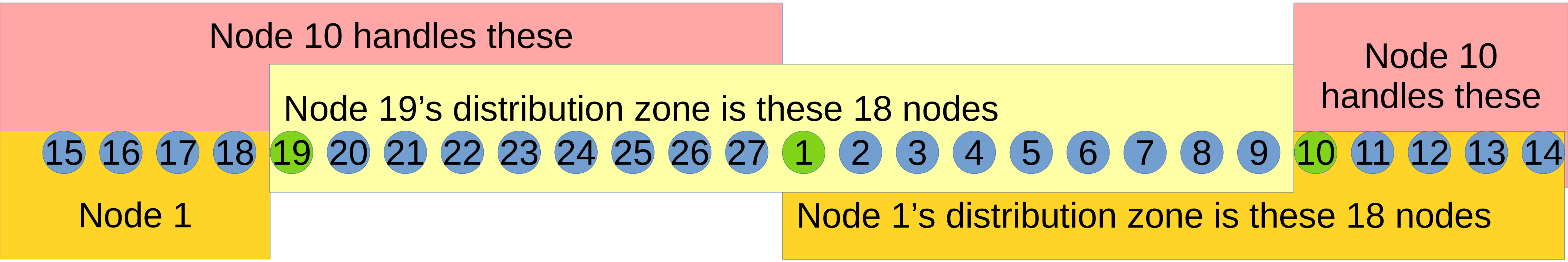
- In fact, <u>all nodes</u> follow this self-demoting behavior.

Side Note: Unneeded rules in V2N

- In networks with highly variant sizes of peer lists Rain Tree has two cases of behavior modification which serve to mitigate the effects of a message source node miscalculating the network size and launching messages with an insufficient or excessive level number. Namely, increment/non-decrement and MaxLevelAdjustment.
- Nodes on Pocket V2N have access to the blockchain source of truth, so these mitigators are not needed in this application.

Why we ACK and adjust on the first layer but nowhere else

Rain Tree splits the target universe into 3 overlapping sets, such that each of the first 3 participants (1, 10 and 19 in the diagram below) is responsible for 2/3rds of the remaining nodes.



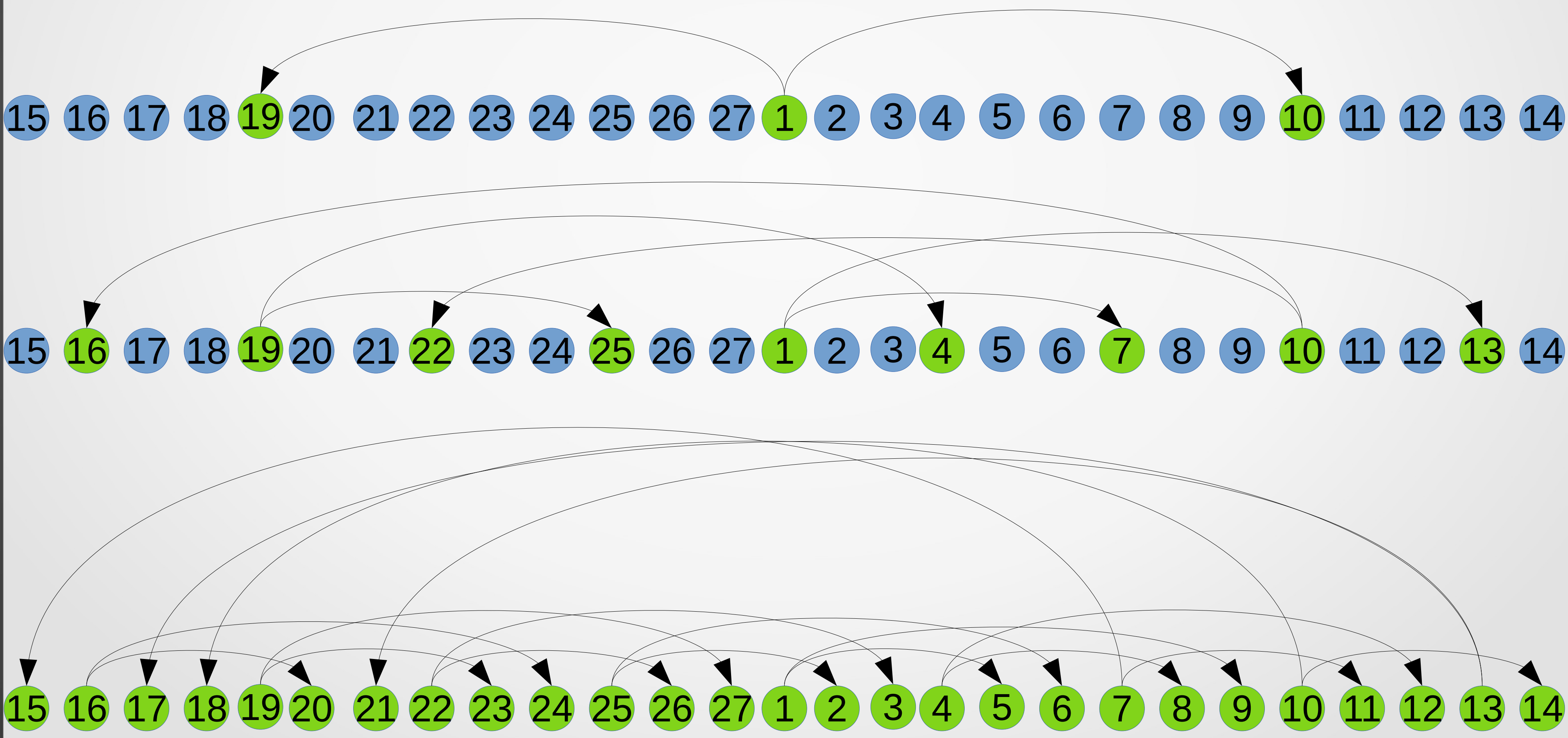
Although we would still have full coverage with the loss of any one zone..
We would not have the full structural redundancy factor of 2 within that subsection.

The clean-up layer and any included redundancy layer would (of course) take care of potential issues.

Ready for some real fun?

Real Rain Tree levels are multiples of 3..
27 = max size of 4 layer network.

Let's work through 3 layers of a 4 layer network. Can you think of anything more fun?



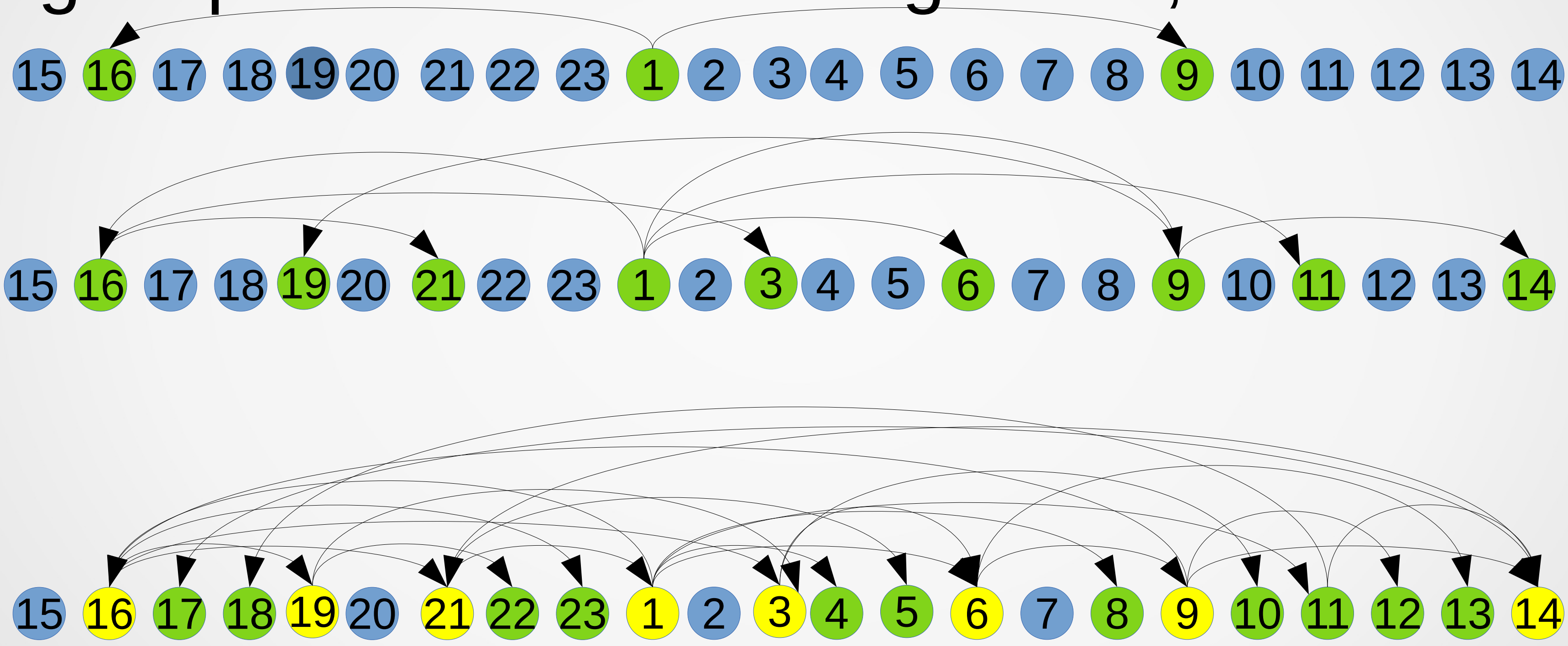
Scaling it up.

- The 4 layer, 27 node network that we have just drawn has the following characteristics:
- Layer 4: Anodes = 1, Comms = 2, ACKs = 2, Ticks = 3
- Layer 3: Anodes = 3, Comms = 6, ACKs = 0, Ticks = 2
- Layer 2: Anodes = 9, Comms = 18, ACKs =0, Ticks = 2
- Redundacy layer 1: Anodes = 27, Comms =27, ACKS =0, Ticks =1
- Clean-up Layer 0: Anodes = 27, IGYWs = 54, ACKS = 54, Ticks =3
- TOTAL(n=27): Comms+IGUWs = 107, ACKs = 61, Ticks = 11
- Redundancy: (Comms+IGUWs) / nodes = <u>3.96</u>
- TOTAL(n=9): Comms+IGUWs = 53, ACKs = 25, Ticks = 10
- Redundancy: (Comms+IGUWs) / nodes = <u>5.88</u>

From perfect to worst case

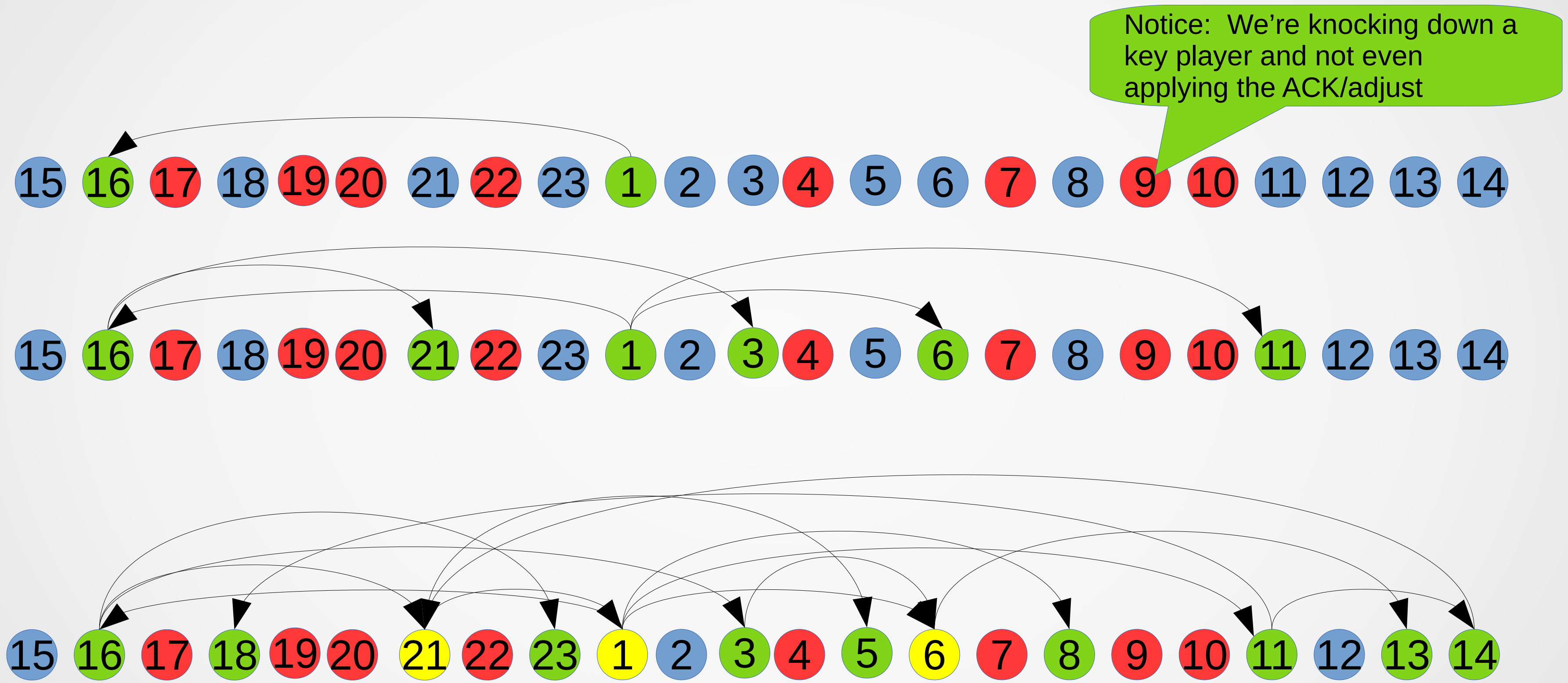
3 x 3 x 3=27 Best Case

Largest prime number in range = 23, Worst Case



Fire Away!!

- Let's knock down 8 of these bad boys. = 35%



15 of 23 nodes on line.
12 of them got the message. (3 got it twice)
Clean up layer fixes it all in 2 ticks.

Simple Math makes life easy.

- How do you determine the nuber of layers in the network?
- topLayer = count factors of 3 then add 1
- =================================
- How to determine targetListSize at spesfic layer?
- targetListSize = (topLayer – currentLayer) x 0.666 x fullListSize
- =================================
- How do nodes determine which 2 nodes to send to?
- Target 1 = Node postion + targetListSize/3 (roll over if needed)
- Target 2 = Node position + targetListSize/1.5 (roll over if needed)

This will scale.

Tripple the nodes… Ticks = +2

Nodes	Comms	ACKs	Ticks
27	107	56	11
81	323	164	13
243	971	488	15
729	2,915	1,460	17
2,187	8,747	4,376	19
6,561	26,243	13,124	21
19,683	78,731	39,368	23
59,049	236,195	118,100	25
177,147	708,587	354,296	27