

Justin Yackoski
CISC861 – Homework #1
20 September, 2004

My Gossip protocol simulation program consists primarily of a 2-dimensional array which holds one “Node” entity for each of the simulated nodes in the 20x20 grid. Each Packet message contains a unique ID, and the x and y position of its original source node, which are constant throughout the life of a Packet. Each Node is aware of its x and y position in the grid to calculate its distance from the source. Each Node also maintains a list of previously seen Packet IDs.

When a node receives a Packet, it first checks the list of previously seen Packets, and if it already saw the Packet, it discards it. Otherwise, the node records that it received a Packet with that ID, and calculates the number of hops between its own distance and the source position of the Packet. If this is less than or equal to K , the packet is re-broadcast unchanged to the Node's 4 neighbors. If it is greater than K , the packet is re-broadcast unchanged to the Node's 4 neighbors with probability P .

Due to the nature of the Parsec simulation, it was necessary to maintain both the x and y position at each Node. In the simulation, time was not considered, so a packet could take 8 hops and reach a node before the same packet reached the node via a geographically shorter route. In a physical world, the delay at each hop resulting from the transmission, propagation, and processing would provide enough delay that we can assume the packet taking the shortest path will nearly always get there first. Additionally, in practice, the x and y position might not be known, so instead a simple TTL equal to K could be set on each Packet, which would be decremented before each re-broadcasting. Once the TTL reached zero, the Nodes would then probabilistically re-broadcast the packet instead of always re-broadcasting it.

It was also necessary to maintain at each Node a list of previous seen Packet IDs to

enforce the rule that duplicate Packets should be ignored. Again, in practice this could be simplified so that we don't need to keep a huge array of IDs. If each Node maintains its own counter used for the Packet ID on Packets it originates, it should be sufficient to assume that other Nodes will receive the Packets from a source Node in roughly the correct order. In this case, each Node only needs an array of the other Nodes in the network containing the highest Packet ID received that originated from each Node.

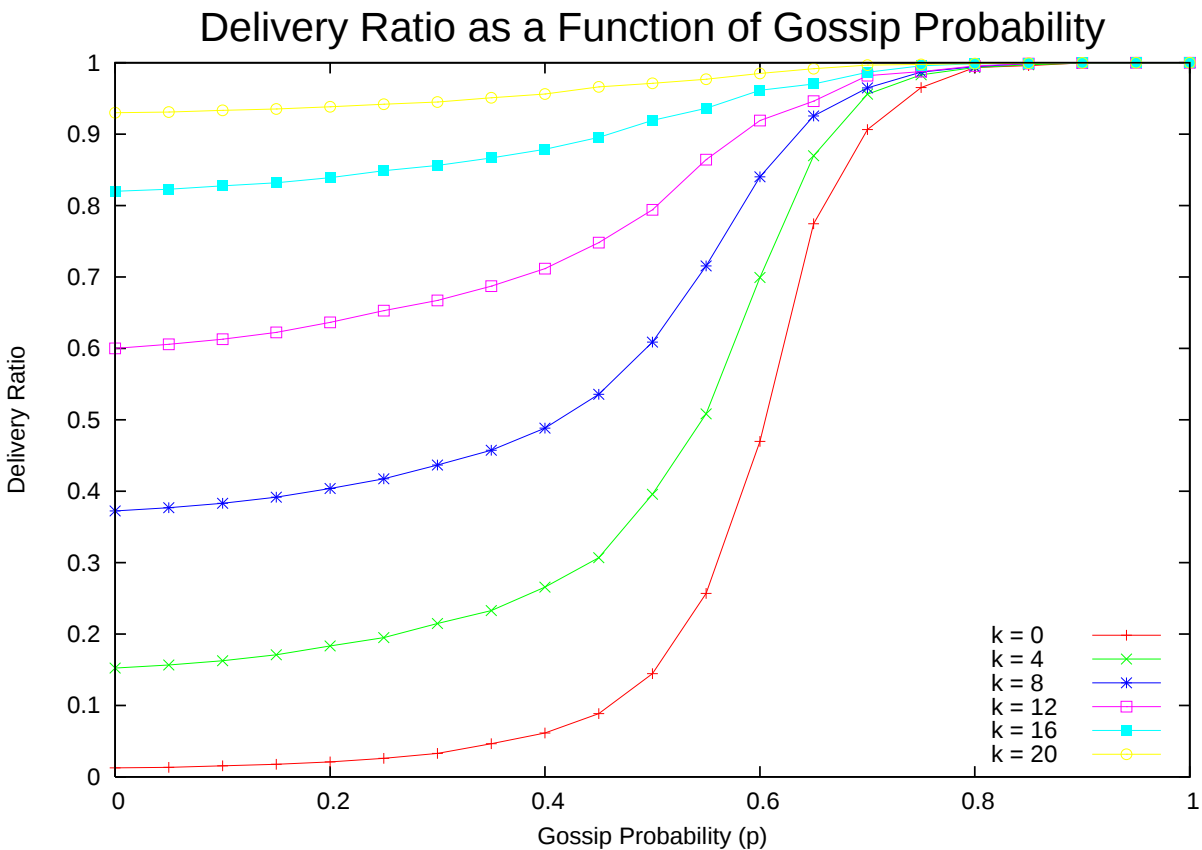


Chart 1 - Delivery Ratio as a Function of Gossip Probability

Chart 1 shows how the Gossip Probability (p) affects the ratio of Nodes in the network which each Packet reaches. For p values less than 0.5, the ratio closely relates to the minimum number of hops the Packet must travel before it is probabilistically re-broadcast (k). Chart 2 shows a better view of the ratios as they approach 1.0. From this chart, between $p = 0.7$ and $p = 0.8$, the lines for $k = 4$ thru 12 are similar, and above $p = 0.8$, k has little impact on the ratio.

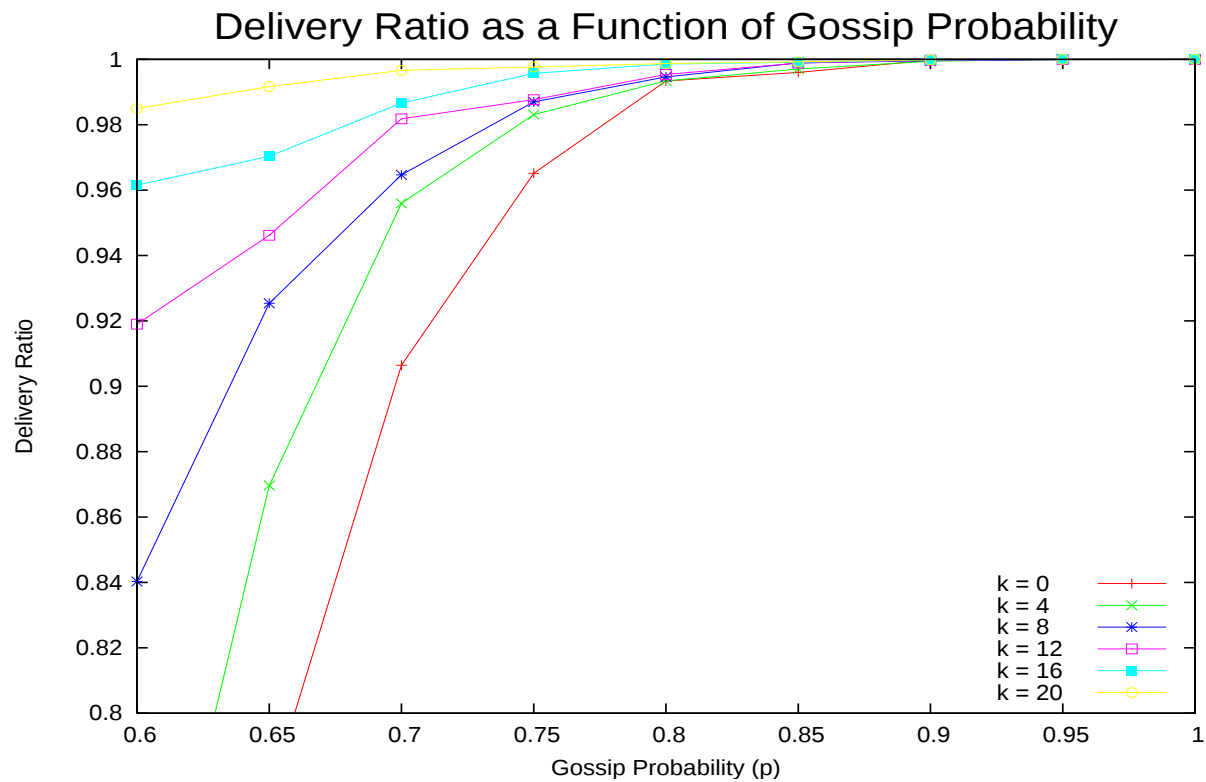


Chart 2 - Delivery Ratio as a Function of Gossip Probability (closer view of ratios > 0.8)

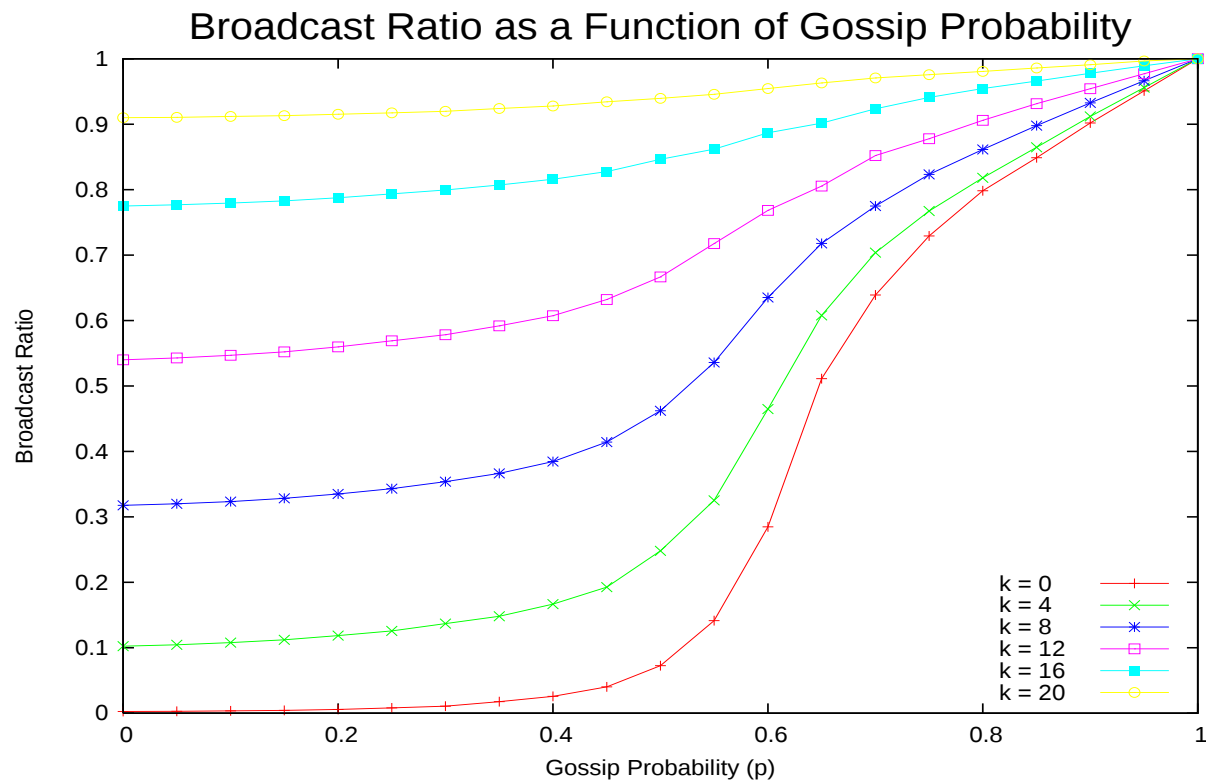


Chart 3 - Broadcast Ratio as a Function of Gossip Probability

Chart 3 shows the broadcast ratio related to p . The interesting aspect of this graph is that, contrary to the delivery ratio, the broadcast ratio increases roughly linearly between $p = 0.8$ and $p = 1$. Thus, when $p > 0.8$, the use of resources to broadcast the additional messages results in no significant increase in the delivery ratio. This data supports the premise of the Gossip protocol, that it can use fewer broadcasts to achieve results similar to flooding. From these graphs, the values which optimize the delivery ratio without wasting broadcasts are in the ranges of $0.7 < p < 0.85$, and $4 < k < 12$.

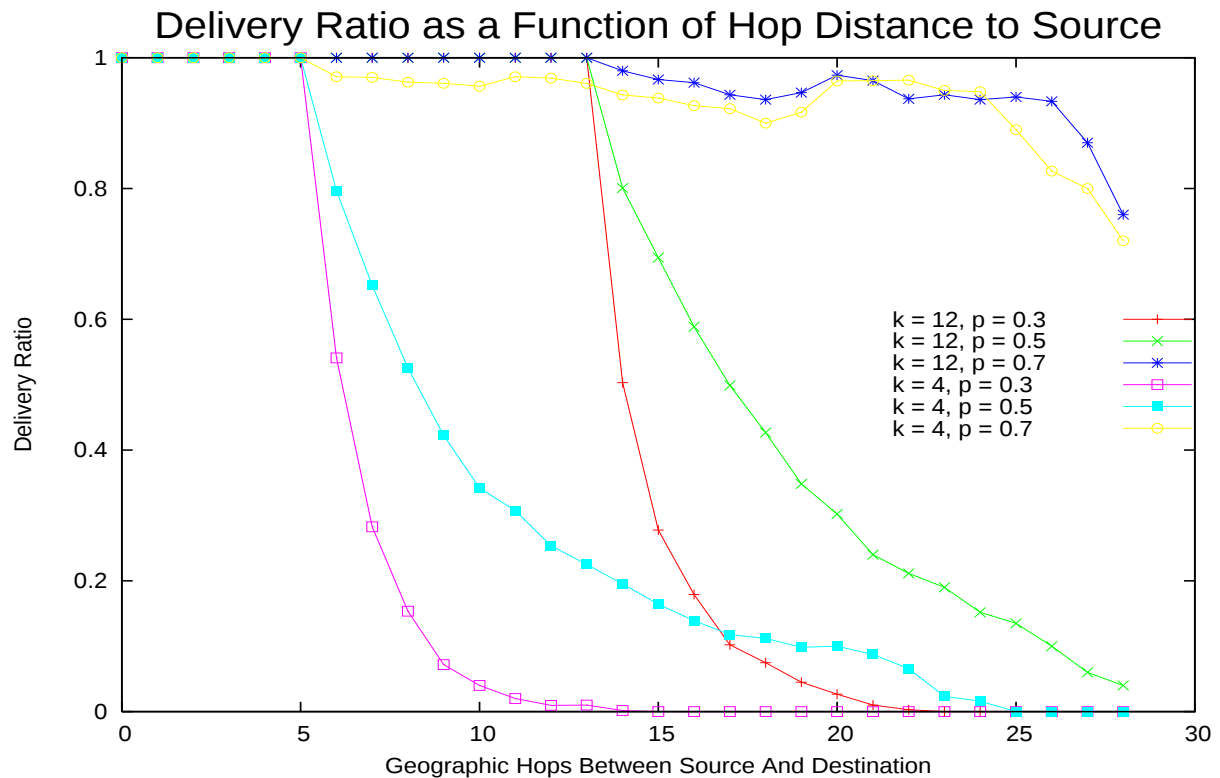


Chart 4 - Delivery Ratio as a Function of Hop Distance to Source

Chart 4 shows the effect of k and p on the delivery rate broken down by the Nodes' physical distance from the source. For values of $p < 0.6$, the delivery rate quickly decays once k hops have been reached. However, for $p = 0.7$, the delivery rates for the two k values (4 and 12) are similar even 25 hops away from the source. The main difference is that between 4 and 12 hops away, when $k = 12$, all nodes 12 or fewer hops from the source receive the Packets. This is

of course the definition of k , and in some applications it may be necessary to ensure a certain number of nodes always receive packets by increasing k . The key observation here is that k should not be increased purely to increase the overall network's delivery ratio, since as Charts 2 and 3 show, increasing k is not as efficient in this respect as increasing p .