Neighbourcast,
a Network Paradigm for MMORPGs

Alexei Bratuhin

September 30, 2008

Contents

Chapter 1

MMORPG

1.1 Definition

MMORPG (Massively Multiplayer Online Player Game) is

- A fully realised, living, breathing online world which exists in its entirety in real-time, allowing players to quest and develop their character, no matter where they are within it at any one time;
- A "shared world" online game, with at least some persistent elements, featuring a large base of otherwise unconnected players.

In an MMORPG, like any RPG, the user controls a character represented by an avatar, which he directs to fight monsters for experience, interact with other characters, acquire items, and so on. MMORPGs have become extremely popular since the wider debut of broadband Internet connections, now with millions of subscribers from hundreds of different countries. Some MMORPGs have as many as a million subscribers.

1.2 History

MMORPGs have their roots in online, text-based adventures, which existed as early as the 1970s. The first real MMORPG, Meridian 59, was released in 1996, but it wasn't until the next year that Ultima Online highly popularized the genre. Both games were played on a pay-by-month basis, as are most modern MMORPGs. The genre surged into popularity throughout the late nineties, finding especially welcoming markets in Taiwan, South Korea, and America. In 1999, Sony Online Entertainment released EverQuest, a popular game to this day. Right around the same time, Ultima Online and Asheron's Call were released, two other MMORPGs that would become extremely popular.

Around 2000, MMORPGs began to attract the attention of academia (psychology and economics) and non-gaming publications. Reactions ranging from praise to distaste are common, with some critics saying that such games turn us into lifeless zombies, and others celebrating the MMORPG as a fascinating new way for us to interact with each other.

In 2001, the market appeared to plateau, causing the cancellation of some MMORPGs in development. But new MMORPGs continue to be released, with dozens of popular games still being sold. One popular selling point is expansion packs, either downloaded from the server as patches or sold in stores. These expansion packs ensure that the virtual worlds in MMORPGs stay fresh and interesting, constantly changing.

Some MMORPGs have developed sophisticated economies with equipment, currency, and characters within the game being exchanged online for real money. This has led to the study of "syntethic economics" and how they relate to real world economies. As MMORPG worlds become increasingly more realistic and entertaining, they will continue to permeate further into the mainstream, attracting both positive and negative reactions from all sides.

1.3 Architecture

- Usually built using client-server architecture, examples World of Warcraft, Everquest.
- Usually thin clients are used - application logic (e.g., player information, player interaction) is implemented on the server side.
- Situation: trade chat within M player group
- Implementation: broadcast or some version of multicast is used, resulting in $O(M^2)$ messages sent through the server.

1.4 Issues

- Maintainance requires sufficient servers and bandwidth. Insufficient resources lead to lags.
  - Commercial grade infrastructure requires hundreds or thousand of servers (millions of subscribers).
- P2P + Tracker (peer-to-peer communication) could work for regulating work load, when not for problems with asymmetrical bandwidth and cheating.
  - DSL solves the asymmetrical bandwidth problem
  - Cheating is always an issue

Chapter 2

Neighbourcast

2.1 Introduction

In MMORPGs the most common information exchanged is the status, the position and the stats information. This information needs to be spread all over the virtual world of MMORPG. But no one from one corner/pole of the map needs to know what's happening on the other corner/pole (unless this is a system or some kind of multicast (group/guild) event).

Therefore information about world changes needs to be spread among the 'neighbours' of the change only.

The most reasonable metric for neighbourhood relation in MMORPG is the sight range of a player.

Assuming information should be sent only within some known region, there arises the problem of finding those neighbours:

- asking the server to check all clients and to return the list of neighbours results in broadcast, if naive approach is applied.
- sorting clients could potentially solve the problem of finding neighbours, if only one ordering is used (not the case, since we have x-y-z, y-x-z, etc.)
- maintaining the neighbourhood graph. But keeping this graph up-to-date in a distributed environment is difficult, since it could require the rebuild of the whole graph
  - Solution: if changing state, client sends messages that have to be forwarded by its neighbours.
  - Problem: since each node doesn't have the complete neighbourhood graph, it would require some predictive capabilities to make the algorithm precise.

2.2 Problem Statement

Develop an algorithm/model for MMORPG that uses position/range information for p2p communication, thus avoiding broadcast for information exchange between nodes that leads to network overloading.

2.3 Definition, Description

Neighbourcast - multicast variation, aims at sending messages to nodes within some predefined range only. It uses epidemiologic style of message forwarding to maintain a state close to consistent.

Nodes within range are called 'neighbours' - and are the only one (except for server) that a node communicates with.

Each node maintains its own list of neighbours that it updates on proceeding messages from neighbours - each message contains 'position' information that gives the node the possibility to check, whether its neighbour has left the node's range or not.

Each node regularly updates its information on tracker server. That reduces the probability of cheating, since tracker server is responsible for holding the client's information and the client itself. That also assures an recovery point for client in case of breakdown.

Procedure:

- send status messages to all nodes from neighbour list with
  - N forwarding hops = 0, if not moving
  - N forwarding hops = 1, if moving
- proceed incoming messages
  - if received message from node that is a neighbour and isn't in neighbourlist ⇒ add node to neighbourlist
  - if received message from node that isn't neighbour and is in neighbourlist ⇒ remove node from neighbourlist - check if graph is still connected
- if not, ask the server to build MST

Note that breakdown of a client during peer-to-peer communication is treated as disconnect. That means, that not proceeded information (messages received, but not yet proceeded and messages not yet received) is then lost. The first client to discover the breakdown initiates MST build procedure to recalculate forwarding routes.

Note that neighbourhood relation is symmetrical. Therefore it may not happen that one of the users sees the other, and the second doesn't see the first one.

Therefore an inconsistency error (situation, when two nodes aren't in neighbourhood relation, although they are in each other's range) may be annoying if not corrected in reasonable time, but is not crucial for the single user's gameplay.

2.4 Problems

A chain of nodes that is (part of) a MST. If N nodes > 4 and nodes move to form a circle, nodes at chain's end don't receive information about each other, although they are in each other's range.

2.5 Enhancements

2.5.1 Logarithmic TTL

Set $N\_forwarding\_hops = log(N) - N\_neighbours$. Logarithmic number of hops makes the probability of occurring of problem situation smaller

- MMORPG node network is unlikely to have a chain structure
  - if consisting of several chains, after MST build the chain structure is unlikely to survive.
- MMORPG node network is likely to have a web-similar structure

2.5.2 Probabilistic Add/Remove Neighbour

Notice that in the step following an MST build an MST is unlikely to change.

- Therefore, removal of distant nodes is more probable, the more time after MST build has passed.
- Therefore, addition of distant nodes is more probable, the more time after MST build has passed.
- $Probability = 1 - 1/max(t - (c - l), 1)$, where
  - c - current step
  - l - last MST build step
  - t - number of steps needed to cross *range* with *speed*

2.6 Alternatives

Notice that neighbourcast is not an exact algorithm. List of the neighbours isn't consistent to reality at each step. Neighbourcast aims at reducing the network load providing somewhat incomplete or erroneous information.

If consistency at each step is crucial (each node at every moment must have correct and consistent information) one considers publish/subscribe systems providing possibility for range queries, e.g., 'Mercury'.

If using publish/subscribe system, a node doesn't maintain a list of neighbours. Instead, it registers its area of interests and then becomes all the messages from events in the area.

Chapter 3

Mercury

3.1 Pub/Sub Introduction

Publish/subscribe is an asynchronous message paradigm where senders (publishers) of messages are not programmed ti send their messages to specific receivers (subscribers). Instead, published messages are categorized into classes, without knowledge of what (if any) subscribers there may be. Subscribers express interest in one or more classes and only receive messages that are of interest, without knowledge of what (if any) publishers there are. This decoupling of publishers and subscribers can allow for greater scalability and a more dynamic network topology.

Types of publish/subscribe architecture:

- Topic-Based Publish/Subscribe - processes exchange information through a set of predefined subjects (topics) which represent many-to-many distinct (and fixed) logical channels.
  - List-Based Publish/Subscribe
  - Broadcast-Based Publish/Subscribe
- Content-Based Publish/Subscribe - are more flexible as subscriptions are related to specific information content and, therefore, each combination of information items can actually be seen as a single dynamic logical channel.

Advantages of Publish/subscribe systems:

- Loosely-coupled
- Scalable (for small- and middle-sized installations)

Disadvantages of Publish/subscribe systems:

- Loosely-coupled - no end-to-end functionality possible
- Poorly scalable for larger installations
- Security problems

  - if using brokers - DoS, DDoS
  - unauthorized publishers (SSH solves the problem at cost of higher network load)
  - unauthorized subscribers (SSH solves the problem at cost of higher network load)

Limitation of topic-based publish/subscribe systems:

- Poor support for range queries (e.g., $1 \leq x \leq 2 \wedge y == "abcd"$ is very inefficient, if topic classification is primarily based on 'z')

3.2 Description

Mercury includes:

- A message routing algorithm that supports range-based lookups within each routing hub in $O(log^2(n)/k)$ when each node maintains k links to other nodes;
- A low-overhead random sampling algorithm that allows each node to create an estimate of system-wide metrics such as data value and load distribution;
- A load-balancing algorithm (which exploits the random sampling algorithm) that ensures that routing load is uniformly distributed across all participating nodes;
- An algorithm for reducing query flooding by estimating how selective each of the predicates in a query are based on past database insertions.

Structure:

- (logically) partition nodes in groups called attribute hub;
- each attribute hub is responsible for a specific attribute;
- first routing hop determines which attribute to route through. The rest of the routing is unidimensional and is based on the values of a single attribute of the data item;
- nodes within each attribute hub are arranged into a circular overlay with each node responsible for a contiguous range of attribute values. Ranges are assigned to nodes during the join process.

3.3 Implemented features

- HubSet
- Hub (Attribute Hub)
- HubNode

- Publication
- Subscription
- Publication TTL
- Load balancing

3.3.1 Differences to Mercury

- Publications are not stored in hubs
- Publications have a TTL attribute that is helpful when deleting (expired / obsolete) publications
- Publications are routed until they (expire / become obsolete)
- Query is proceeded on each attribute hub, an intersection of found subscriptions is then built

3.4 Not Implemented features

- Small-world network structure
- Sampling for load balancing
- Routing in $O(log^2(n)/k)$

Chapter 4

Test and Test Results

To check the grade of inconsistency of the proposed algorithm and also to evaluate the algorithm in comparison to the most naive approach - broadcast - a number of test has been launched. The tests were launched with different parameters (number of nodes, size of game field, speed, range, etc.). As results, the number of inconsistency errors, the number of 'build MST'-calls and the number of messages sent overall was computed and stored for analysis.

Since calculating of inconsistencies required strong concurrency control (for timestamping purposes) it was decided to launch number of iterative test, meaning that nodes ran corresponding neighbourcast routine synchronously. Due to limitations of existing plotting systems and low computational power following 3 parameters fixed:

- Field size = 1000x1000
- # Nodes = 100
- # Steps = 1000

Results of the test are shown on the graphic for each algorithm:

- Neigbourcast
- Neighbourcast $\wedge$ 2.5.1. (Neighbourcast_CWA)
- Neighbourcast $\wedge$ 2.5.2.

Diagram Legend:

- topleft graphic shows test results for the Neighbourcast/Broadcast ratio, meaning ratio between number of messages sent and number of messages that would be sent if using broadcast;
- topright graphic shows test results for the Neighbourcast/Broadcast ratio in interval [0:1], meaning test that performed better (in terms of network load), that Broadcast would has performed;

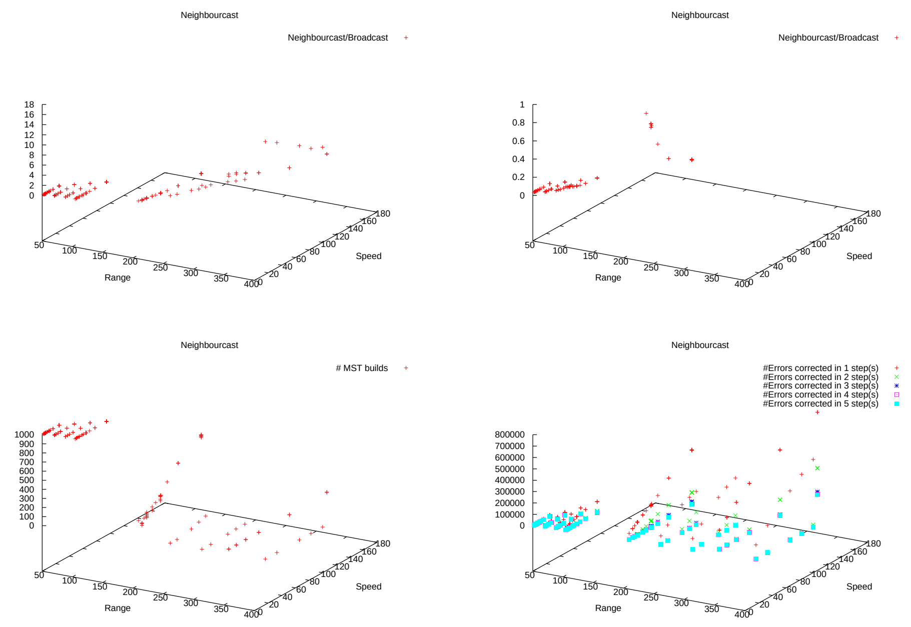

Figure 4.1: Test Statistics: Neighbourcast

- bottomleft graphic shows the number of MST build operations performed by the server. Since building an MST in complete graph has complexity $O(N^2)$ the less such operations performed the better;
- bottomright graphic shows the number of inconsistencies fixed in 1, 2, ... ,5 steps, where an inconsistency is the situation when two nodes aren't in neighbourhood relation, although they are in each other's range.

Let us now discuss the test results:

- First of all, some tests end with Neighbourcast/Broadcast ratio > 1. This effect is caused by forwarding messages (as specified in neighbourcast routine) in a very dence neighbourhood graph;
- Second, most tests have a good Neighbourcast/Broadcast ratio - < 0.1 - for reasonable values of *Speed* and *Range* ($Speed << Range << Field$)
- Third, both "Neighbourcast" and "Neighbourcast with Logarithmic TTL" require large amount of 'build MST' operations - appr. each step and appr. each 2nd step, correspondingly. "Neighbourcast with Probabilistic Add/Remove Neighbour" scores slightly better.
- Forth, both "Neighbourcast" and "Neighbourcast with Logarithmic TTL" have large amount of inconsistencies, corrected in 1 or 2 steps (which makes the algorithm follow the past), whereas "Neighbourcast with Probabilistic Add/Remove Neighbour" results in very few inconsistencies (which makes the algorithm closely follow the reality).

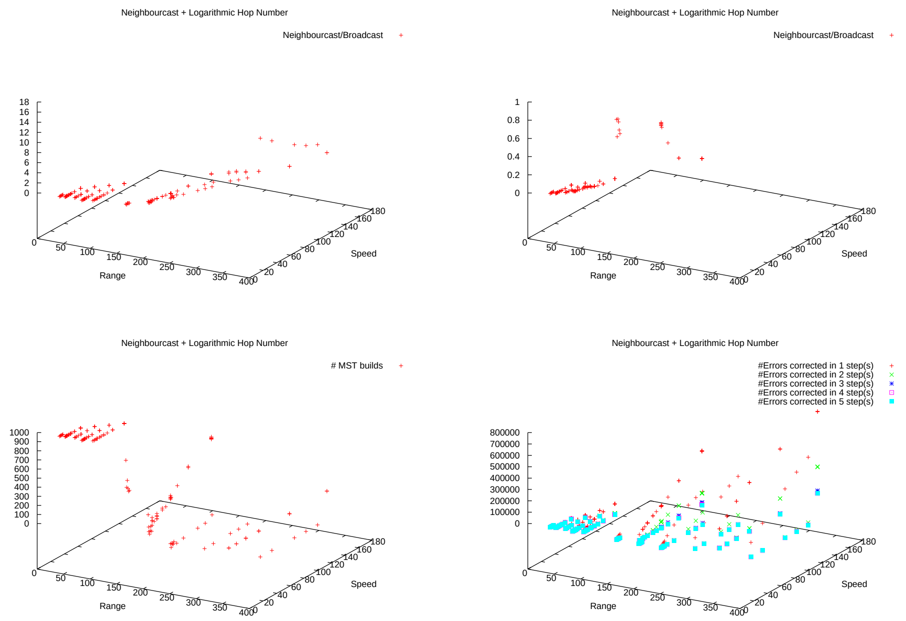

Figure 4.2: Test Statistics: Neighbourcast with Logarithmic TTL

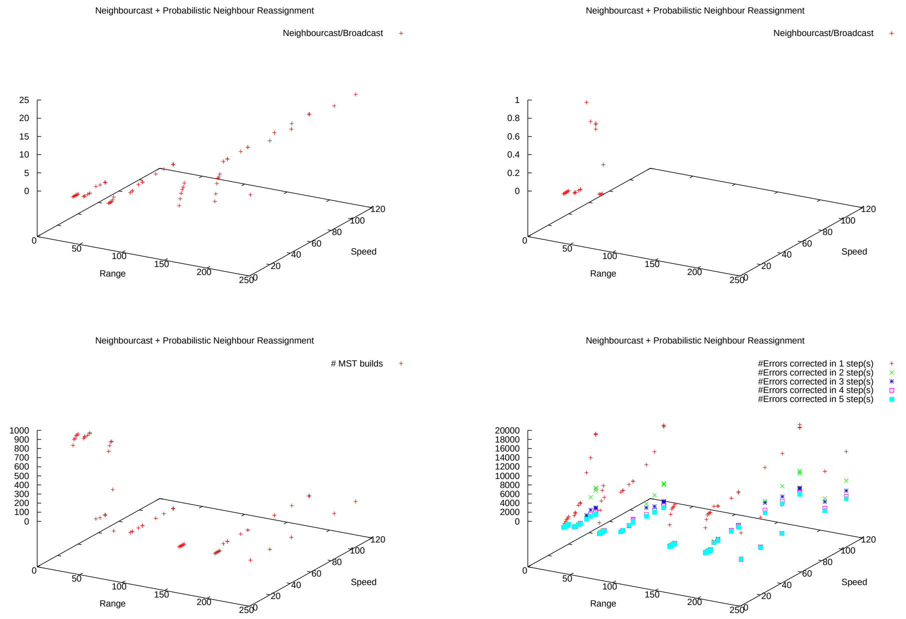

Figure 4.3: Test Statistics: Neighbourcast with Probabilistic Add/Remove Neighbour

Chapter 5

Conclusion

Summarizing results we conclude that

- using Neighbourcast for reasonable values of Range and Speed ($Speed << Range << Field$) efficiently reduces the network load at costs of inconsistencies in neighbourhood graph most of which corrected in at most 5 steps. However a unaffordably huge number of MST computations is made;
- using Neighbourcast with Logarithmic TTL reduces number of errors and speeds up their correction, as well as reduces number of MST computations made at very small costs;
- Using Neighbourcast with Probabilistic Add/Remove Neighbour shows good results in some parameter constellations and reduces number of inconsistencies to acceptable level. However the chosen probablity function doesn't have a continuous effect on test results. It is too optimistic in most testcases and is not applicable in general case. More sophisticated probability function may be needed for probabilistical enhancement of Neighbourcast.

Relating to MMORPGs, Neighbourcast can be applied for sparse populated areas, such as "outtown" terrains. Neighbourcast is definitely not suited for "intown" areas, especially marketplaces.

Finally, it must be again stressed that Neighbourcast (also with enhancements) is not an 'exact' algorithm, which means it doesn't guarantee consistent state of the neighbourhood graph at every moment. If maintaining such consistent state is important, Publish/Subscribe systems should be considered a good choice. Using Mercury protocol should theoretically result in $O(n * log^2(n)/k)$ complexity for sending messages to neighbours only, compared to $O(n^2)$ using naive Broadcast ($n * log^2(n)/n^2 \simeq 0.05, n \simeq 100$).

Chapter 6

Literature and Links

- Mercury
  - http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.61.1608
  - www.sigcomm.org/sigcomm2004/papers/p625-bharambe1.pdf
  - www.cs.cmu.edu/ ashu/talks/sigcomm2004.ppt
  - www.cs.uoi.gr/ pitoura/courses/p2p/slides/tsotsos8.ppt
  - www.cs.cornell.edu/people/egs/cs615-spring07/mercury.pdf
  - http://doi.acm.org/10.1145/1030194.1015507
- Rest
  - http://en.wikipedia.org/wiki/Publish/subscribe
  - http://msdn.microsoft.com/en-us/library/ms978603.aspx
  - http://www.wisegeek.com/what-is-a-mmorpg.htm

Chapter 7

Appendix A - Test Results As Table

Table 7.1: Neighbourcast; Ratio

Speed/Range	50	66	83	100	200	250	300	333	400
2	0.028	-	-	-	-	-	-	-	-
3	0.027	-	-	-	-	-	-	-	-
4	0.028	0.046	-	0.141	-	-	-	-	-
5	0.027	-	0.075	-	-	-	-	-	-
6	-	0.044	-	0.134	-	-	-	-	-
7	0.027	-	-	-	-	-	-	-	-
8	0.026	-	0.071	0.124	1.666	-	-	-	-
9	-	0.039	-	-	-	-	-	-	-
10	0.026	-	-	0.119	-	3.683	-	-	-
11	-	-	0.062	-	-	-	-	-	-
12	0.025	-	-	-	-	-	6.651	-	-
13	-	0.036	-	-	1.461	-	-	-	-
14	-	-	-	0.104	-	-	-	-	-
16	0.025	-	0.053	0.094	1.412	-	-	-	17.262
20	-	-	-	0.080	1.372	3.205	-	-	-
22	-	0.032	-	-	-	-	-	8.903	-
25	0.025	-	-	0.070	-	-	5.798	-	-
27	-	-	0.043	-	-	-	-	-	-
28	-	-	-	-	1.104	-	-	-	-
33	-	0.030	-	0.057	0.950	-	-	8.068	15.775
40	-	-	-	-	0.804	-	-	-	-
41	-	-	0.037	-	0.781	2.379	-	-	-
47	-	-	-	-	-	-	-	7.197	-
50	-	-	-	0.045	0.542	-	4.518	-	-
52	-	-	-	-	-	1.833	-	-	-
62	-	-	-	-	-	1.479	3.779	-	-
66	-	-	-	-	0.315	-	-	5.831	12.680
75	-	-	-	-	-	-	3.103	-	-
83	-	-	-	-	-	-	-	-	10.868
100	-	-	-	-	0.161	-	-	-	9.843
111	-	-	-	-	-	-	-	3.446	-
166	-	-	-	-	-	-	-	2.045	-

Table 7.2: Neighbourcast; Number of MST Builds

Speed/Range	50	66	83	100	200	250	300	333	400
2	1000	-	-	-	-	-	-	-	-
3	1000	-	-	-	-	-	-	-	-
4	1000	1000	-	1000	-	-	-	-	-
5	1000	-	1000	-	-	-	-	-	-
6	-	1000	-	1000	-	-	-	-	-
7	1000	-	-	-	-	-	-	-	-
8	1000	-	1000	1000	209	-	-	-	-
9	-	1000	-	-	-	-	-	-	-
10	1000	-	-	1000	-	15	-	-	-
11	-	-	1000	-	-	-	-	-	-
12	1000	-	-	-	-	-	2	-	-
13	-	1000	-	-	155	-	-	-	-
14	-	-	-	1000	-	-	-	-	-
16	1000	-	1000	1000	203	-	-	-	1
20	-	-	-	1000	224	13	-	-	-
22	-	1000	-	-	-	-	-	3	-
25	1000	-	-	1000	-	-	1	-	-
27	-	-	1000	-	-	-	-	-	-
28	-	-	-	-	276	-	-	-	-
33	-	1000	-	1000	302	-	-	1	1
40	-	-	-	-	352	-	-	-	-
41	-	-	1000	-	349	41	-	-	-
47	-	-	-	-	-	-	-	1	-
50	-	-	-	1000	459	-	9	-	-
52	-	-	-	-	-	71	-	-	-
62	-	-	-	-	-	96	16	-	-
66	-	-	-	-	600	-	-	2	2
75	-	-	-	-	-	-	15	-	-
83	-	-	-	-	-	-	-	-	2
100	-	-	-	-	760	-	-	-	2
111	-	-	-	-	-	-	-	7	-
166	-	-	-	-	-	-	-	23	-

Table 7.3: Neighbourcast; Number of Inconsistencies x100

Speed/Range	50	66	83	100	200	250	300	333	400
2	15/0/0/0/0/	-	-	-	-	-	-	-	-
3	21/0/0/0/0/	-	-	-	-	-	-	-	-
4	29/0/0/0/0/	65/2/0/0/0/	-	161/6/0/0/0/	-	-	-	-	-
5	29/0/0/0/0/	-	121/4/0/0/0/	-	-	-	-	-	-
6	-	85/3/0/0/0/	-	231/8/0/0/0/	-	-	-	-	-
7	41/1/0/0/0/	-	-	-	-	-	-	-	-
8	44/1/0/0/0/	-	195/9/0/0/0/	289/11/0/0/0/	538/15/4/3/2/	-	-	-	-
9	-	118/4/0/0/0/	-	-	-	-	-	-	-
10	54/2/0/0/0/	-	-	339/14/1/0/0/	-	754/15/4/3/3/	-	-	-
11	-	-	246/11/0/0/0/	-	-	-	-	-	-
12	52/1/0/0/0/	-	-	-	-	-	955/13/5/4/7/	-	-
13	-	148/7/0/0/0/	-	-	784/35/6/4/2/	-	-	-	-
14	-	-	-	485/29/2/0/0/	-	-	-	-	-
16	62/2/0/0/0/	-	321/19/2/0/0/	529/34/3/0/0/	956/59/8/3/1/	-	-	-	1254/14/10/4/14/
20	-	-	-	635/49/5/0/0/	1137/82/9/3/1/	1378/46/6/4/4/	-	-	-
22	-	203/13/1/0/0/	-	-	-	-	-	1698/17/7/8/7/	-
25	85/3/0/0/0/	-	-	720/60/9/1/0/	-	-	1798/31/8/6/9/	-	-
27	-	-	456/38/5/0/0/	-	-	-	-	-	-
28	-	-	-	-	1527/208/25/7/2/	-	-	-	-
33	-	246/16/1/0/0/	-	818/81/13/2/0/	1710/342/44/9/2/	-	-	2419/34/10/10/12/	2389/15/9/8/13/
40	-	-	-	-	1970/540/83/13/2/	-	-	-	-
41	-	-	523/52/7/1/0/	-	2043/589/103/15/2/	2456/316/30/6/3/	-	-	-
47	-	-	-	-	-	-	-	3297/74/10/11/13/	-
50	-	-	-	938/110/21/5/1/	2474/850/176/29/5/	-	3293/220/22/6/3/	-	-
52	-	-	-	-	-	2740/672/64/8/2/	-	-	-
62	-	-	-	-	-	2928/1099/162/14/2/	3805/476/20/6/1/	-	-
66	-	-	-	-	3469/1100/220/60/17/	-	-	4333/268/20/11/3/	4326/71/17/15/13/
75	-	-	-	-	-	-	4171/884/58/10/5/	-	-
83	-	-	-	-	-	-	-	-	5212/133/26/10/8/
100	-	-	-	-	4810/1064/252/86/30/	-	-	-	5953/232/24/11/8/
111	-	-	-	-	-	-	-	5745/1399/92/10/4/	-
166	-	-	-	-	-	-	-	7241/2343/236/16/2/	-

Table 7.4: Neighbourcast with Logarithmic TTL; Ratio

Speed/Range	41	50	66	83	100	166	200	208	250	300	333	400
1	0.033	-	-	0.114	-	0.984	-	2.347	3.759	-	-	-
2	0.036	0.042	-	0.102	-	0.790	-	1.942	3.814	-	-	-
3	0.034	0.040	-	0.096	-	0.980	-	1.914	3.960	-	-	-
4	0.034	0.041	0.060	0.094	0.154	0.944	-	2.249	4.092	-	-	-
5	0.034	0.040	-	0.092	-	0.851	-	1.879	3.656	-	-	-
6	0.034	-	0.058	0.088	0.144	0.807	-	1.903	3.565	-	-	-
7	-	0.039	-	-	-	-	-	-	-	-	-	-
8	-	0.039	-	0.087	0.138	-	1.567	-	-	-	-	-
9	-	-	0.054	-	-	-	-	-	-	-	-	-
10	-	0.038	-	-	0.129	-	-	-	3.753	-	-	-
11	-	-	-	0.081	-	-	-	-	-	-	-	-
12	-	0.037	-	-	-	-	-	-	-	6.612	-	-
13	-	-	0.051	-	-	-	1.526	-	-	-	-	-
14	-	-	-	-	0.121	-	-	-	-	-	-	-
16	-	0.037	-	0.071	0.106	-	1.382	-	-	-	-	17.440
20	-	-	-	-	0.098	-	1.325	-	3.118	-	-	-
22	-	-	0.045	-	-	-	-	-	-	-	8.996	-
25	-	0.037	-	-	0.084	-	-	-	-	6.137	-	-
27	-	-	-	0.055	-	-	-	-	-	-	-	-
28	-	-	-	-	-	-	1.105	-	-	-	-	-
33	-	-	0.042	-	0.071	-	1.003	-	-	-	8.147	15.644
40	-	-	-	-	-	-	0.791	-	-	-	-	-
41	-	-	-	0.048	-	-	0.767	-	2.304	-	-	-
47	-	-	-	-	-	-	-	-	-	-	7.245	-
50	-	-	-	-	0.057	-	0.558	-	-	4.483	-	-
52	-	-	-	-	-	-	-	-	1.840	-	-	-
62	-	-	-	-	-	-	-	-	1.470	3.794	-	-
66	-	-	-	-	-	-	0.326	-	-	-	5.834	12.403
75	-	-	-	-	-	-	-	-	-	3.213	-	-
83	-	-	-	-	-	-	-	-	-	-	-	10.956
100	-	-	-	-	-	-	0.179	-	-	-	-	9.863
111	-	-	-	-	-	-	-	-	-	-	3.428	-
166	-	-	-	-	-	-	-	-	-	-	1.999	-

Table 7.5: Neighbourcast with Logarithmic TTL; Number of MST Builds

Speed/Range	41	50	66	83	100	166	200	208	250	300	333	400
1	1000	-	-	1000	-	872	-	319	46	-	-	-
2	1000	1000	-	1000	-	570	-	128	1	-	-	-
3	1000	1000	-	1000	-	642	-	97	44	-	-	-
4	1000	1000	1000	1000	1000	551	-	131	27	-	-	-
5	1000	1000	-	1000	-	514	-	174	25	-	-	-
6	1000	-	1000	1000	1000	521	-	117	47	-	-	-
7	-	1000	-	-	-	-	-	-	-	-	-	-
8	-	1000	-	1000	1000	-	146	-	-	-	-	-
9	-	-	1000	-	-	-	-	-	-	-	-	-
10	-	1000	-	-	1000	-	-	-	14	-	-	-
11	-	-	-	1000	-	-	-	-	-	-	-	-
12	-	1000	-	-	-	-	-	-	-	1	-	-
13	-	-	1000	-	-	-	196	-	-	-	-	-
14	-	-	-	-	1000	-	-	-	-	-	-	-
16	-	1000	-	1000	1000	-	175	-	-	-	-	1
20	-	-	-	-	1000	-	185	-	29	-	-	-
22	-	-	1000	-	-	-	-	-	-	-	1	-
25	-	1000	-	-	1000	-	-	-	-	2	-	-
27	-	-	-	1000	-	-	-	-	-	-	-	-
28	-	-	-	-	-	-	250	-	-	-	-	-
33	-	-	1000	-	1000	-	264	-	-	-	2	1
40	-	-	-	-	-	-	317	-	-	-	-	-
41	-	-	-	1000	-	-	335	-	50	-	-	-
47	-	-	-	-	-	-	-	-	-	-	7	-
50	-	-	-	-	1000	-	425	-	-	8	-	-
52	-	-	-	-	-	-	-	-	59	-	-	-
62	-	-	-	-	-	-	-	-	109	6	-	-
66	-	-	-	-	-	-	570	-	-	-	5	2
75	-	-	-	-	-	-	-	-	-	8	-	-
83	-	-	-	-	-	-	-	-	-	-	-	2
100	-	-	-	-	-	-	741	-	-	-	-	2
111	-	-	-	-	-	-	-	-	-	-	10	-
166	-	-	-	-	-	-	-	-	-	-	25	-

Table 7.6: Neighbourcast with Logarithmic TTL; Number of Inconsistencies x100

Speed/Range	41	50	66	83	100	166	200	208	250	300	333
1	0/0/0/0/0/	-	-	21/2/0/0/0/	-	58/5/2/0/0/	-	71/3/2/0/0/	77/4/5/0/1/	-	-
2	2/0/0/0/0/	6/0/0/0/0/	-	34/2/0/0/0/	-	107/6/2/0/0/	-	129/4/3/0/1/	149/5/5/1/2/	-	-
3	3/0/0/0/0/	7/0/0/0/0/	-	50/2/0/0/0/	-	177/6/2/1/0/	-	197/6/3/0/0/	229/5/5/1/1/	-	-
4	4/0/0/0/0/	12/0/0/0/0/	34/1/0/0/0/	70/2/0/0/0/	140/5/0/0/0/	231/6/2/1/0/	-	275/6/3/1/0/	301/6/6/1/2/	-	-
5	6/0/0/0/0/	13/0/0/0/0/	-	89/3/0/0/0/	-	271/10/3/1/0/	-	319/9/4/1/1/	361/7/5/1/2/	-	-
6	8/0/0/0/0/	-	53/1/0/0/0/	96/3/0/0/0/	184/6/0/0/0/	327/11/2/1/0/	-	386/11/4/1/1/	429/10/5/1/1/	-	-
7	-	20/0/0/0/0/	-	-	-	-	-	-	-	-	-
8	-	25/0/0/0/0/	-	151/5/0/0/0/	252/10/0/0/0/	-	510/16/4/1/1/	-	-	-	-
9	-	-	78/3/0/0/0/	-	-	-	-	-	-	-	-
10	-	32/1/0/0/0/	-	-	287/12/0/0/0/	-	-	-	769/15/5/2/1/	-	-
11	-	-	-	212/10/0/0/0/	-	-	-	-	-	-	-
12	-	38/1/0/0/0/	-	-	-	-	-	-	-	958/14/9/2/4/	-
13	-	-	119/5/0/0/0/	-	-	-	799/34/7/3/0/	-	-	-	-
14	-	-	-	-	462/24/1/0/0/	-	-	-	-	-	-
16	-	48/1/0/0/0/	-	314/19/1/0/0/	496/30/2/0/0/	-	925/57/10/3/1/	-	-	-	-
20	-	-	-	-	602/44/4/0/0/	-	1120/86/9/3/1/	-	1382/45/7/3/2/	-	-
22	-	-	182/11/0/0/0/	-	-	-	-	-	-	-	1694/20/12/5/7/
25	-	71/2/0/0/0/	-	-	693/55/7/0/0/	-	-	-	-	1837/30/10/5/5/	-
27	-	-	-	403/31/4/0/0/	-	-	-	-	-	-	-
28	-	-	-	-	-	-	1456/215/28/9/1/	-	-	-	-
33	-	-	233/15/1/0/0/	-	820/82/13/2/0/	-	1686/344/46/10/1/	-	-	-	2439/34/11/5/5/
40	-	-	-	-	-	-	1903/543/89/11/1/	-	-	-	-
41	-	-	-	493/46/6/0/0/	-	-	1969/602/97/13/2/	-	2422/324/25/10/6/	-	-
47	-	-	-	-	-	-	-	-	-	-	3290/87/18/7/6/
50	-	-	-	-	926/107/20/4/1/	-	2394/850/185/28/4/	-	-	3274/218/15/9/5/	-
52	-	-	-	-	-	-	-	-	2720/685/59/8/2/	-	-
62	-	-	-	-	-	-	-	-	2904/1082/161/14/2/	3840/473/22/9/3/	-
66	-	-	-	-	-	-	3317/1108/226/57/15/	-	-	-	4285/275/21/7/6/
75	-	-	-	-	-	-	-	-	-	4253/854/43/9/2/	-
83	-	-	-	-	-	-	-	-	-	-	-
100	-	-	-	-	-	-	4767/1076/245/79/29/	-	-	-	-
111	-	-	-	-	-	-	-	-	-	-	5754/1394/85/6/3/
166	-	-	-	-	-	-	-	-	-	-	7178/2365/243/18/2

Table 7.7: Neighbourcast with Probabilistic Add/Remove Neighbour; Ratio

Speed/Range	41	50	83	100	166	200	208	250
1	0.034	-	0.106	-	3.053	-	6.037	9.647
2	0.035	-	0.097	-	4.761	-	7.933	-
3	0.035	-	0.088	-	6.133	-	10.618	-
4	0.036	0.040	0.092	-	6.996	-	11.924	-
5	0.035	0.040	0.401	-	7.519	-	12.092	-
6	0.036	-	1.049	-	8.463	-	12.752	-
8	-	0.042	-	2.667	-	-	-	-
10	-	0.042	-	2.841	-	-	-	-
16	-	0.961	-	3.661	-	14.283	-	-
20	-	0.726	-	3.692	-	14.351	-	21.531
25	-	0.675	-	-	-	-	-	-
33	-	-	-	3.871	-	14.376	-	-
40	-	-	-	-	-	14.456	-	-
41	-	-	-	3.960	-	-	-	21.462
50	-	-	-	3.858	-	-	-	-
66	-	-	-	-	-	14.401	-	-
83	-	-	-	-	-	14.227	-	21.276
100	-	-	-	-	-	14.149	-	-
104	-	-	-	-	-	-	-	21.110

Table 7.8: Neighbourcast with Probabilistic Add/Remove Neighbour; Number of MST Builds

Speed/Range	41	50	83	100	166	200	208	250
1	900	-	908	-	1	-	1	1
2	957	-	963	-	1	-	1	-
3	965	-	998	-	1	-	1	-
4	988	971	997	-	1	-	1	-
5	988	990	461	-	1	-	1	-
6	993	-	1	-	1	-	1	-
8	-	983	-	1	-	-	-	-
10	-	996	-	1	-	-	-	-
16	-	14	-	1	-	1	-	-
20	-	1	-	1	-	1	-	1
25	-	1	-	-	-	-	-	-
33	-	-	-	1	-	1	-	-
40	-	-	-	-	-	1	-	-
41	-	-	-	1	-	-	-	1
50	-	-	-	1	-	-	-	-
66	-	-	-	-	-	1	-	-
83	-	-	-	-	-	1	-	1
100	-	-	-	-	-	1	-	-
104	-	-	-	-	-	-	-	1

Table 7.9: Neighbourcast with Probabilistic Add/Remove Neighbour; Number of Inconsistencies x100

Speed/Range	41	50	83	100	166	200	208	250
1	3/0/0/0/0/	-	24/1/0/0/0/	-	48/2/1/1/0/	-	56/2/2/1/1/	60/2/3/1/2/
2	7/0/0/0/0/	-	50/3/0/0/0/	-	70/2/1/0/0/	-	73/2/2/1/1/	-
3	9/0/0/0/0/	-	69/2/0/0/0/	-	80/2/1/1/0/	-	82/2/2/1/1/	-
4	13/0/0/0/0/	24/0/0/0/0/	91/3/0/0/0/	-	83/2/1/1/0/	-	84/2/2/1/1/	-
5	14/0/0/0/0/	28/0/0/0/0/	100/3/0/0/0/	-	83/2/1/1/0/	-	84/2/2/1/2/	-
6	17/0/0/0/0/	-	73/1/1/1/1/	-	87/3/2/1/0/	-	84/3/2/1/1/	-
8	-	42/1/0/0/0/	-	88/2/1/1/2/	-	-	-	-
10	-	44/1/0/0/0/	-	90/2/2/2/2/	-	-	-	-
16	-	104/11/9/2/1/	-	95/4/4/5/6/	-	82/5/3/2/1/	-	-
20	-	132/45/17/7/3/	-	97/7/8/12/7/	-	77/5/4/3/2/	-	70/6/5/5/3/
25	-	178/58/14/7/1/	-	-	-	-	-	-
33	-	-	-	117/31/23/8/4/	-	78/8/7/8/6/	-	-
40	-	-	-	-	-	81/9/8/9/6/	-	-
41	-	-	-	136/40/16/6/3/	-	-	-	71/9/8/9/8/
50	-	-	-	180/52/14/6/2/	-	-	-	-
66	-	-	-	-	-	105/31/27/11/6/	-	-
83	-	-	-	-	-	114/43/19/8/4/	-	92/33/26/11/6/
100	-	-	-	-	-	151/52/17/8/4/	-	-
104	-	-	-	-	-	-	-	109/46/24/11/6/