The CyberBeanie Toy Routing System:

A Resource-saving Network-Layer

6.033 Design-Project 1
Recitation: Snoeren / Freedman TR12

Wolfgang K. Gatterbauer
wolf@mit.edu

March 22, 2001

Abstract

This paper explores the idea of creating a toy routing system that enables furry balls to exchange “emotions” with their pre-determined symmetric partner in a conglomeration of such toys by using a short-range radio and the concept of forwarding packets. The author recommends a network-layer design which avoids routing tables by sending messages to all nodes in the network. In order to avoid resending of packets, each node keeps track of recently sent messages in a FIFO-memory, implemented with a purge pointer algorithm. With regard to the specifications, the analysis compares the recommended and alternative solutions by addressing tradeoffs between necessary RAM, packet size, minimum sending intervals, and overall network traffic.

1 Introduction

A fictitious toy manufacturer asks us to help in the design of the new CyberBeanie toy (CB). This gadget is a furry ball about three inches across with the key feature that each CB forms a “friendship” with exactly one other CB. A pair of friendly CBs reflect each other’s simulated “emotions,“ meaning that if one is heated up in the palm of a hand, an LED on the other turns on. The marketing department thinks this feature will be a hit with 5-year-old school children.

Each CB contains an inexpensive embedded controller CPU, a short-range radio, a given link- and end-to-end-layer (e2e). Two CBs within radio range can directly communicate.

Our task is to design the algorithms and protocols for the network-layer that allow communication over longer distances within a group of CBs by having them forward packets for each other.

2 System Specifications and Design Criteria

Specification of the existing equipment:

- Each CB comes from the factory pre-programmed with its unique 32-bit ID and the 32-bit ID of its symmetric friend that are known by the e2e-layer.
- Whereas the ROM for storing instructions is unlimited the RAM usable by the network-layer is only 256 bytes, in which all tables, variables and prepared packets for transmission have to be stored.
- The CB radio has a range of 5 feet. Each unit can send only one packet at any time. It can, however, receive simultaneous transmissions without interference and can queue incoming and outgoing messages in the several hundred bytes comprising memory of the link-layer. The radios are full-duplex and the useable data rate for the network-layer is 1000 bits per second.
- The network-layer has to provide two routines to the e2e-layer – one that is called for initialization when a CB powers up, and one by which a 16 bit temperature is handed over that should be delivered to the unit’s friend. The e2e-layer itself is capable with of dealing with duplicate, miss-ordered and lost messages, and provides a e2e_deliver()-function, by which the network-layer conveys information from the unit’s friend.
- The link-layer provides a link_send()-function by which the network-layer passes the content and the length of a packet. In exchange, the link-layer hands up a package to the network-layer by calling a net_deliver()-function with the same arguments.

Intended use of the system:

- The children tend to sit in rows of a square classroom. In such an arrangement, each child is just under five feet from the children on either side and in front and back, but about seven feet from the nearest diagonally-adjacent children. However, the system should not collapse into chaos if, for example, a unit has more than 4 neighbors, or if the units are not arranged in a perfect grid. The more robust the system, the better.
- The children sit still during each one-hour class, but then move around before sitting down in a possibly different arrangement for the next class.
- The inter-message period will be a fixed parameter. Shorter would be better, since it would make the toy more interactive.
- A destination might not be present in the classroom.

Questions to consider in the design:

- How much memory does the design consume as a function of the number of children?
- How often should temperature messages be sent? How does this period depend on the number of total CBs in the network?
- How long does it take the system to stabilize after each pause?
- How does the system work in a classroom of 25 children?
- Can the hardware capabilities (radio speed and memory size) be reduced if the CBs were required to work correctly in groups of up to 100 and could take up to 20 seconds to stabilize.

Resulting main considerations for the design:

- The shorter the packet size, the more packets per time each CB can send.
- The more detailed a route between two friends is known and either contained in the packet or remembered by each node, the fewer total packets are transferred in the network during each cycle.
- The memory is very small, which makes it difficult to keep detailed routing tables.
- A CB cannot determine from which neighbor a message was transferred, except for the case that he message contains this information.
- Transmission cannot be directed beyond the network-layers.

3 Recommended Design Solution

I start with two definitions for simplifying reasoning:

Cycle: Minimum time frame within which every CB in the network has a chance to exactly once send its temperature to his friend and no more packets are circulating in the net thereafter.

Period: Time it takes a CB to transfer exactly one package.

Inter-message period: Time interval at which a CB sends out messages.

I recommend a broadcast solution, in which every CB receives and forwards every message except the one intended for itself (In one-dimensional arrangements, CBs might actually receive fewer messages). No “routing tables” are created, but “remembering tables” of sent messages to avoid a CB to resend messages that a neighbor transmits in one of a few periods after the message had already passed.

This approach needs no time for setting up any network parameters and the stabilization period can therefore be equaled to a simple cycle.

3.1 Information Propagation

Figures 1, 3 and 5 show examples of how information might propagate from node A to B. Circles, loops pose no problem as the information propagates simultaneously into all directions.

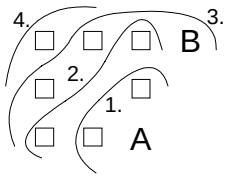


Figure 1: Information Propagation

Messages can generally not arrive out of sequence. It might only happen in the very unlikely case that 1.) the topology contains holes, 2.) one out of two possible routes to the destination is considerably more congested than the other, 3.) sending of consecutive messages happens within only a couple of periods.

3.2 Protocols and Code

Figure 2 shows the structure of the 50-bit-packet. It contains the 4-Byte-ID of the intended destination, the 2-byte temperature and a 2-bit-sequence number which makes a packet distinguishable from the previous and following 3 messages originating from the same source. A CB can hereby distinguish between a new message and one that it has

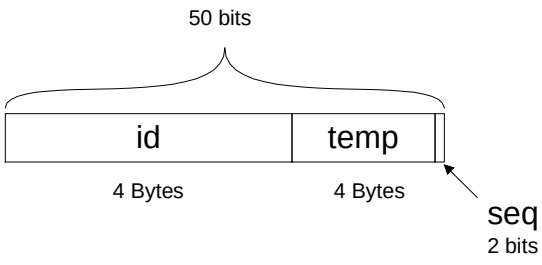


Figure 2: The network-protocol

already been forwarded a couple of periods ago.

The fact that the protocol uses a 2 bit-sequence number instead of 1-bit makes the system more robust. A packet might get lost and after not receiving one message from a certain destination, a CB might mistake the next received message for the prior received message and decide not to forward it. With the 2-bit-sequence it would take 3 lost messages until the network-layer would illegitimately discard a message.

The code, listed in appendix A, contains an algorithm that allows only one entry in the table for each ID. If it finds an entry with the same ID, but a different and therefore older sequence number, it overwrites it with the new value. This uniqueness for IDs is important for a network with a small number of CBs. Otherwise, a CB might store all possible 4 sequence numbers for one ID in the course of 4 cycles and decide not to forward any other messages from this source anymore. Because packets can be assumed not to arrive out of sequence, as reasoned in 3.1., this algorithm is robust.

A CB receives a message intended for itself more than once (except for the case that it has only one neighbor or is the only functioning connection between two or more groups of CBs). At every such arrival it delivers the contained temperature to the e2e-layer, which, according to the specifications, is able to deal with duplicate messages. It does not forward a message intended for itself.

3.3 Layout of Memory Usage

Figure 3 shows the layout of the recently sent message table that can hold up to 48 such entries. A purge pointer algorithm remembers the least recently written entry and overwrites it with a new incoming message in a network with more than 50 CBs (48 + 2 for itself and friend) [1].

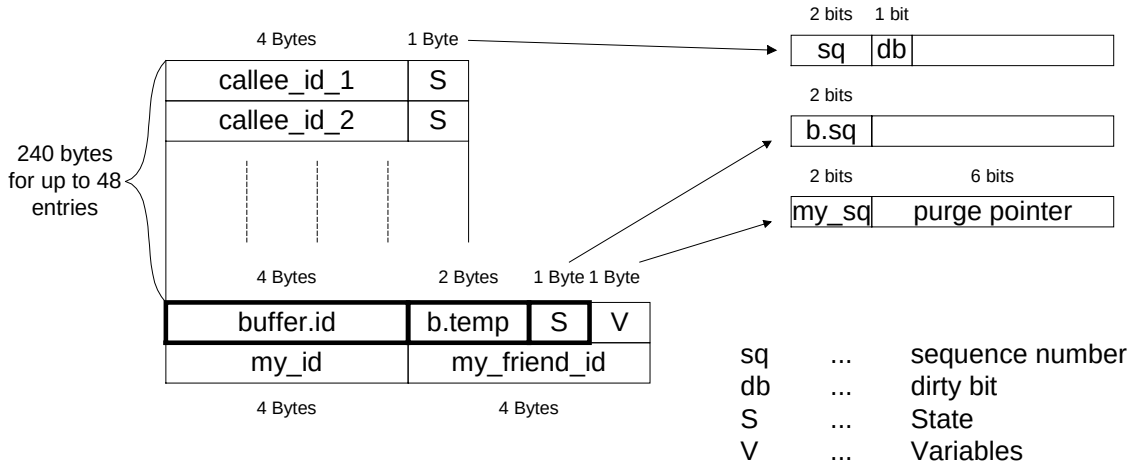


Figure 3: Usage of the 256 bytes RAM

RAM poses no restrictions on scalability in this system. Except for the case of an overloaded network, in which the restrictions of 3.4. are not regarded, no way could be conceived in which a message would be resent by one neighbor after a CB has received 48 other messages (not to confuse with 48 periods which would be even less likely !).

3.4 Performance and Limits

According to the specifications, the 1000 bit-rate of the transceiver can be entirely used by the network-layer. Ergo, a CB can transmit up to 20 (= 1000 bit per sec / 50 bits per message) messages per second, and a period takes exactly 50ms. Using the results of Appendix B, a cycle will take less than 1.5 times the number of nodes of periods. To be on the safe side, this reasoning leaves us with the following easy formula:

$$\textit{Minimum Intermessage Period} = 1.5 \cdot n \cdot 50ms$$

For a group of 25 CBs the inter-message period can be set to less than 2 seconds and for a group of 100 CBs to 7.5 seconds.

Put in another way, a system that sends messages in an interval of 20 seconds can support at least up to 266 CBs for the worst case (and highly likely up to 400 CBs in a square matrix).

The system shows no adverse effects if a destination is not present in the network.

3.5 Possibility of Reducing Hardware Resources

Memory for the network-layer can definitely be reduced. The size of the RAM is entirely determined by the maximal number of neighbors plus the number of buffered outgoing messages in the neighbors' link layers any CB might have.

Reducing the RAM to 128 bytes still leaves space for up to 22 such entries (128 – 16 = 112 bytes / 5 bytes per entry = 22.4 ~ 22), which should generally be enough for a matrix as shown in figure 4a. Figure 4b shows that if children gather, e.g. during pauses, this system might be overloaded.

Figure 4c exemplifies the idea that message could only come back after more than one hop in the case that something is changed in the topology within one cycle.

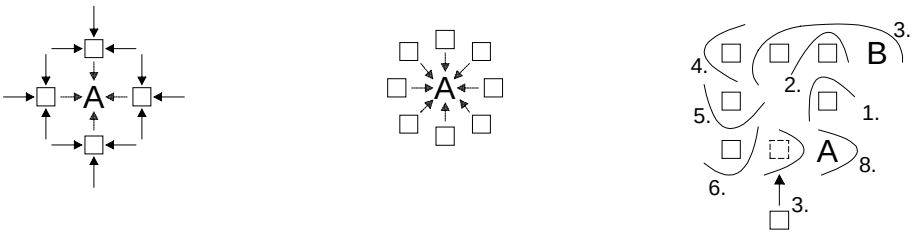


Figure 4a-c: Constraints on the RAM

Appendix C proposes an optimized 64-byte solution with different RAM layout and improved purge pointer algorithm that might even permit the memory to be reduced to only 64 bytes without losing too much of robustness.

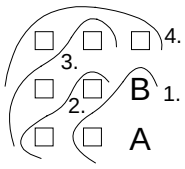
The transmission rate could be decreased for a systems that is supposed to support up to 100 CBs with a resending interval of 20 seconds. The formula derived in 3.4. implies a factor of 20 / 7.5 for the worst case and a factor of 20 / 5 for the reduced requirement of permitting only square matrixes which amounts to an optimized transmission rate of 375 bits per sec (= 1000 / (20/7.5)) and respectively 250 bits per second (= 1000 / (20/5))

4 Alternative Design Solutions

4.1 A more Sophisticated and almost better Solution

In the case that friends are very close to each other in the classroom, a broadcasting system in which every CB sends out every message – except for the one intended for itself – might actually not be very resource-saving.

In an arrangement such as figure x, a message from A gets for warded even after the 4th period although A's friend B receives the message already in the 1st period. Adding a Time-To-Live (TTL) sequence can reduce traffic and save battery power.



Figur 5

The link layer provides the length of the package together with the package itself in net_deliver(). This number can be used for variable length messages and for distinguishing between initializing and common messages. At initialization, A sends out a package with the minimum length of 48 bits, setting temp to 0. Every node receives this packet and adds the value 1 to temp before resending it. In figure 5, B receives this message from different neighbors with temp values of 0, 2 and 3. It remembers the smallest number (0) which from now on becomes the TTL sequence added to the in 3. described 50-bit-messages sent by B. Each node reduces the TTL in an incoming message by one and

forwards it unless it receives a TTL of 0. In order to avoid resending of an initialization message not containing the sequence number, the CBs network algorithms might provide a special routine for storing them in the RAM.

The reduction in common traffic heavily depends on the topology and ranges from nothing in a ring topology where friends are exactly opposed to each other to a significant reduction to only one sent message (!) per CB in the case that all CBs have their friends as direct neighbors. In the latter case, the number of total common messages in a network of *n* CBs is only *n* compared to my previous solution with $n \cdot (n-1)$.

For randomly distributed friends in a square matrix such in figure 8, I estimate that the total traffic can be reduced by an average factor of 4. Appendix D shows an idea how this might actually be proven.

In order to become tolerable to changes within the arrangement of the children, the system has to repeat the initialization when it does not hear from its friend during one inter-message period . It can assume that his friend has moved further away and cannot be reached with the current TTL-setting. In order to save bandwidth, the system has to recognize a situation in which the friend actually moves closer. Therefore, it should send out regular initialization messages in order to work on the optimum. These messages, on the other hand, increase the total traffic except for the case that the network-layer disregards orders from the e2e-layer during initialization.

Because the description of the e2e-layer says that “the inter-message period will be a fixed parameter, set at the factory to whatever value [we] determine”, and our system should be robust and able to handle the worst case topology with a given number of students, I decided not to pursue this solution. It would actually increase the delay due to the increased variable packet size.

One way to make the system more interactive using this idea is to implement an adaptive e2e-layer that automatically adopt to different measured round trips. The e2e-layer and network-layer designer would have to work together and change the interface . But this solution would make the network usage less fair, as close friends would be able to communicate more often with each other. Only the in 4.1.1 proposed simple interaction between e2e- and link-layer would not be able to ensure an equal usage of the network.

As a conclusion, this advanced design would not actually make the CBs more interactive under the given specifications. It might probably neither permit the optimal 64-Byte-RAM solution from Appendix C. The only advantage would be an on average reduced traffic and therefore saving of battery power.

4.2 Further Reducing the Traffic on the Network

In order to further reduce the number of packets sent through the whole network, a system has to be conceived in which CBs remember preferred routes.

As in the system of 4.1., a initialization message will be sent out that is characterized by a unique bit-length mod 32-number. Instead of increasing the number in the temperature, however, every hop adds its ID to the message before forwarding it. The addressee again receives the packet from different neighbors and remembers the route with the least number of hops.

From then on, a CB includes the IDs of this remembered nodes into an originating common message, and a CB that receives a message will only forward it if its ID is included in the packet.

This system is not more powerful than my proposed solution. In a compact 2D-topology, the number of CBs that forward each message on average increases with only $\sqrt{n}$, as compared to *n* in solution 3. But on the other hand, the length of each message would increase as well with $\sqrt{n}$, which results in total again to n. In addition the system would still have to use the purge pointer algorithm to avoid resending messages.

As a conclusion, the reduction of traffic with this solution heavily depends on the network topology. Calibrating the system for the worst case would actually make it less interactive than the originally presented system. This system might, however, become more powerful with increasing locality of reference (a tendency of friends to sit close to each other) and an e2e-layer that permits adjustable sending periods.

4.3 Reducing the Traffic by storing IDs of "permitted" Messages

A way to avoid the increasing packet length of solution 4.2. would be to store, at each node, the IDs of the messages that are most efficiently delivered by passing at this node.

After having received an initialization message from its friend A and having determined the shortest path as described in 4.2., B sends out another specifically marked message which contains the IDs of the CBs that have delivered the message by the shortest number of hops. Receiving this "marker-message", the addressed CBs save A's ID in their memory. Common messages, who have the same format as described in 3.2., get only forwarded when the destination ID is contained in the memory.

This solution would probably become the most powerful of all mentioned solutions when either storage capacity were not restricted to 256 bytes or the number of allowed CBs were far less than 100. The system still needs the purge algorithm to avoid resending messages. Using the 64 byte-minimum design from Appendix C, 256 – 64 = 188 bytes. This space has to be used for the IDs of "permitted" messages, the IDs of alternatively traveled routes during initialization, and some pointer to address this data, which leaves space for 188 / 4 = 47 – 1 for the pointer and some timing variable during initialization = 46 IDs in total. Because the amount of necessary memory depends on a couple of circumstances and timing issues the system cannot be guaranteed to work to work for over 25 CBs.

For 25 CBs it would actually work as it would permit the worst case scenario that I can think of – a ring topology with special arrangements of friends, a node that is part of all

shortest routes between friends and an initialization message from a friend at a time a table of "permitted" messages is already established: 11 + 12 IDs for alternative paths + $2 \cdot 11$ IDs of permitted messages = 45 IDs.

The number of hops each packet travels can as well be approximated by $\sqrt{n}$ as in 4.3. In contrary to 4.3., however, the size of common packets would constantly be 50 bit as in 3, which means that this design is the best scalable solution in regard to interactivity. The average acceptable inter-message period would be as well proportional to $\sqrt{n}$, as opposed to *n* in the recommended solution.

As conclusion, this solution would be the most optimized one with regard to bandwidth usage, and it would be definitely the best one, if the constriction of RAM-space could be re-negotiated and the inter-message period were adaptive.

4.4 An Attempt to get rid of the Purge-Pointer-Algorithm

In order to get rid of the purge pointer algorithm – which I consider to be essential for any broadcasting solution – , one could conceive a system in which the IDs of e.g. the last 4 hops by which a message has traveled are saved. If all passed hops were saved, one could immediately change to the better solution of 4.2. as otherwise the increase in packet length were not balanced by any reduction in the number of forwarded packets.

In this system, however, a node would forward the same message received by different neighbors, several times. As well, it would not be able to tolerate wholes in a matrix. In the example of figure 6, where one child left for the bathroom, a package sent by A and destined for B would circulate forever.

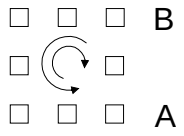


Figure 6

4.5 Arguments against Hard Modularity and Layering

According to generally accepted principles of network design, each layer is forbidden to interpret the data received from the layer above [2]. This ideology permits complex systems to remain simple and comprehensible, and the thoughts of the original designer more easily reproducible.

In a system with very limited resources, which is not intended to be reused for other purposes, however, it might be advantageous to deviate from these common design principles and look for solutions in which the different layers actually work together to share information and resources at the cost of modularity. This strategy might be acceptable if it leaves the final product better off in the eyes of the customer and can help gain acceptance more quickly (idea misused from [3]).

4.5.1 Adaptive Inter-Message Periods

The solution of 4.3. permits adaptive inter-message periods implemented entirely in the e2e-layer. Because the route between friends is determined after initialization, a e2e-layer could be proposed that sends a message back to his friend immediately after receiving a message. The system would be self-regulating as a congestion in the link-layers on the path would lead to slower reaction times.

In my recommended solution of 3., the e2e-layer is not able to achieve a reasonable adaptive load functionality by its own. In a situation such as in figure 5, A and B would happily decrease their inter-message-periods, not realizing that they jam the whole network.

Alternatively, the resending time in the end-to-end-layer can be set rather small and discretion given to the link layer to discard requests from its own machine when the buffer for outgoing messages becomes too big. This means that two different calls of link_send would have to be implemented to distinguish between forwarded and originating messages. In order to avoid the e2e-layer to mistakenly interpret an increased response time as a situation where its friend is not present, an additional reasonably high time constant would have to be found and implanted in the e2e-layer.

I think that some form of load variability would make sense for the system; the toys could become more interactive on average and more fun to the children.

4.5.2 Redundant information in the 2 byte Temperature

According to the specification, the end-to-end software is capable of dealing with duplicate and miss-ordered messages. The only way for it to do so, is to send encrypted sequence information in the 2 byte temperature. Otherwise a temperature that can have 2^16 = 65,536 values would make absolutely no sense from an economic point of view. A 1-byte temperature information would still permit 2^8 = 256 different states which should be sufficient if the information is only used to operate a LED on a CB's friend.

If the network-layer designer got initiated into the protocols of the end-to-end layer, sequence numbers, and therefore bandwidth and RAM could be saved by using this information.

I repeat, such a pulling down of modularity concepts will excite a vehement protest from any experienced systems designer. No rules, however, should ever be established that cannot be questioned.

5 Conclusion

This paper has shown several possible designs of the CB-network-layer. The proposed solution exceeds all performance criteria and permits to even reduce the memory space and the transfer rate for the given specifications. The system is easily scalable without restrictions on the RAM and demands a minimum inter-message period that linearly increases with the number of total CBs in the group. The design needs no initialization messages, is tolerant to all different adverse constellations of CBs flexible to sudden changes, uses multiple paths to transmit a message, and is in all extremely robust and predictable.

The paper as well proposes slight modifications to the specifications which would significantly increase the performance for increasing maximum numbers of CBs, and points out that in that case another solution might achieve higher performance.

Acknowledgments

I feel obliged to thank a couple of people who, in the course of this semester, have taken the time to help me catch up with some basic computer architecture concepts that I was lacking two months ago: Serena Chan, my tutor Danny Park, 6.004-TA Nathan Williams, 6.033-TA Michael Freedman, Cedric Hutchings, and others. Concerning this paper, however, I have developed the ideas of all mentioned network concepts without any help of other people.

References

[1] T. Kaehler, "Virtual Memory for an Object-Oriented Language," BYTE Magazine, Aug., p. 386, 1981.

[2] J. H. Saltzer, M. F. Kaashoek, Topics in the Engineering of Computer Systems. MIT 6.033 class notes, draft release 1.11, Feb., p. 4-36, 2001.

[3] R. Gabriel, “LISP: good news, bad news, how to win BIG,” AI Expert, June, pp. 33-35, 1991.

Appendix A – Code

```
initialize_network (int x, int y)
   {
   my_id = x;
   my_friend_id = y;
   my_seq = 0;                // Initialize my sequence number to 0.
   for (int i=0; i<48; i++)   // For the FIFO-purge-pointer-algorithm
      Callee[i].db = 0;       // to work correctly, all 48 dirty
   purge_pointer = 0;         // bits and the purge pointer have to
   }                          // be initialized with 0.



net_deliver (int packet, int length)
   {
   if (packet.id == my_id)          // this packet is for us and will
      e2e_deliver (packet.temp)     // not be re-sent
   else
      if ((packet.id != my_friend_id)
         &&  (not packet_in_memory (packet.id, packet.seq)) )
         {

// We avoid to resend packets that originate from our own CB
// or are already in the table. The other ones are forwarded
// and saved in a new entry in the table, using the purge pointer
// algorithm below or, or by overwriting an old entry of the same ID
// with the new sequence number in the function packet_in_memory().
// The purge pointer algorithm looks for an entry with a dirty
// bit of 0, sets the dirty bits of all visited entries to 0, and
// create a new entry with dirty bit = 1 for the message.

         while (*purge_pointer.db != 0
            {
            *purge_pointer.db = 0;
            purge_pointer = (purge_pointer + 1) % 47;
            }
         *purge_pointer.id = packet.id;
         *purge_pointer.seq = packet.seq;
         *purge_pointer.db = 1;
         buffer.id = packet.id;
         buffer.temp = packet.temp;
         buffer.seq = packet.seq;
         link_send (buffer, 50);
         }
   }



net_send (int temp)
   {
   buffer.id = my_friend_id;   // Before link_send() gets called, the
   buffer.temp = temp;         // values are written in the buffer.
   buffer.seq = my_seq;
   link_send (buffer, 50);
   my_seq = (my_seq + 1) % 4;  // The sequence number has to be changed
   }                           // in preparation for the next temp.
```

```
int packet_in_memory (int packet.id, int packet.seq))
   {
   buffer.temp = 0  // borrows the for the moment unused buffer memory
                    // to store bit variable before the function returns

   for (int i = 0; i<48; i++)
      if (callee[i].id == packet.id)
         if (callee[i].seq == packet.seq)
            buffer.temp = 1
         else
            {
            callee[i].seq = packet.seq;
            callee[i].db = 1;
            buffer.id = packet.id;
            buffer.temp = packet.temp;
            buffer.seq = packet.seq;
            link_send (buffer, 50);
            }
   return buffer.temp;
   }
```

Appendix B – The Maximum Periods Issue

The worst scenario – the case in which a packet takes the longest time to arrive at its destination – happens in a perfect loop or a 1-dimensional arrangement as given in figure 7:

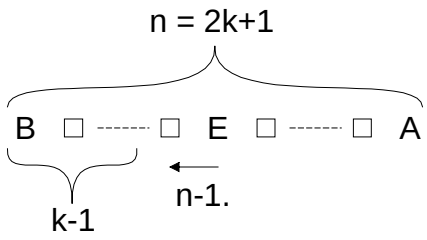


Figure 7: Worst Case Scenario

In the case of an uneven number n = 2k+1 of CBs, E might send A as the last of all outgoing messages of one cycle – at period n-1. The message from A would than take another k-1 periods to arrive at B which adds up to

$$\sum = n-1+k-1 = 3k-1 = \frac{3}{2}(n-1)-1.$$

The same formula holds for an even number of n+1. For the examples of 25 (100) CBs in the net, the formula gives us a maximum cycle time of 35 (146) periods. If n increases the number tends to approach $1.5 \cdot n$:

$$\lim_{n\to\infty} \frac{\frac{3}{2}(n-1)-1}{n} = \frac{3}{2}$$

For topologies which approach a coherent 2 dimensional shape such as a square or even a circle, the maximum periods time comes close to n-1.

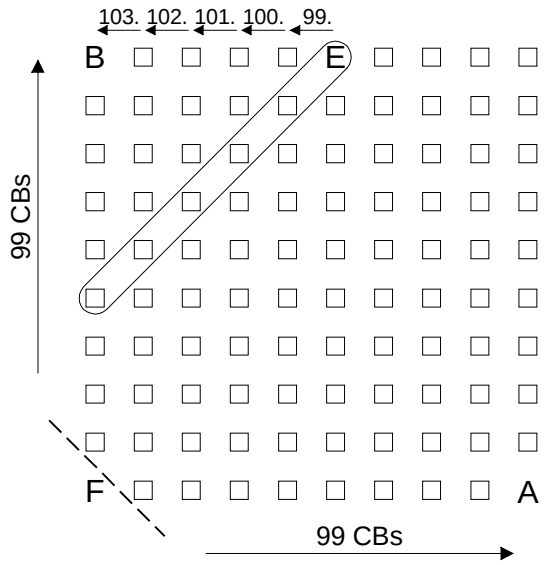


Figure 8: Square Matrix

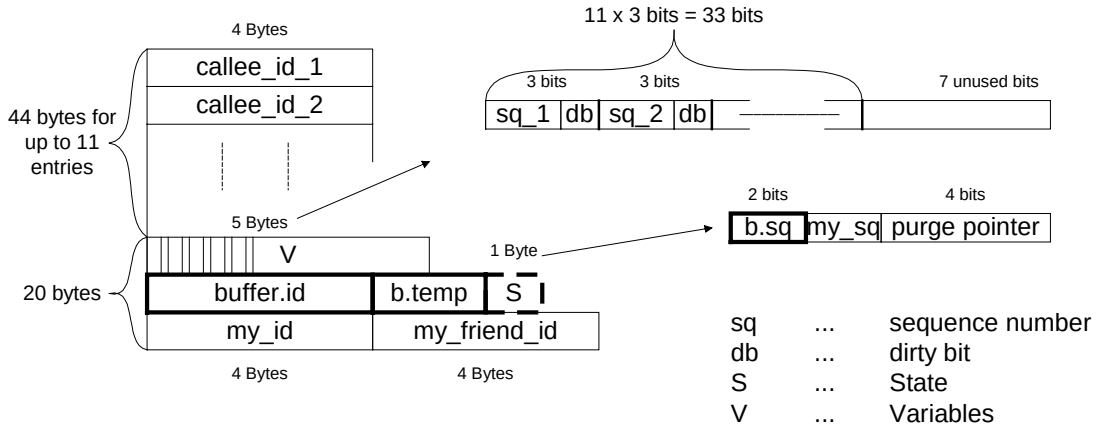
The worst scenario for a square of $10 \cdot 10$ CBs is given in figure 8. A sends out a message destined for B at approximately the same time as all the other CBs. The buffers of the CB's link layers quickly fill up with outgoing messages and the link layers impose an arbitrary order in the queue of messages. In the unlikely case that E and all other CBs on the marked diagonal send out A's message as the last within this cycle (=100–1), B receives A's message only after 103 instead of 99 periods. In order to be possible F has to be E's friend, as only then E would be able to send A's message as last one. The same applies to the CB on the other side of the diagonal.

This might be as well happen if A simply sends out its message as the last one of one cycle and within one period (50ms) after the first CB sent out its message. As this case depends on timing issues, it might be called an unlikely race condition.

Because we want to establish a guaranteed systems response for n CBs, we should calculate with the simplified maximum periods time of $1.5 \cdot n$ for one cycle.

Appendix C – 64 byte RAM Solution for 11 Entries

Figure 9 shows a more compact structure for a 64-byte-RAM that permits to hold 11 entries. Without any changes to the algorithms this solution works well except for the case that for any CB the number of neighbors plus the number of buffered outgoing messages in the neighbors' link layers exceeds 10.



Figur 9: Usage of the 256 bytes RAM

By slightly changing the packet_in_memory () procedure as shown below, the system becomes more robust. Without trying to prove it, I estimate that this 64-byte-RAM broadcast solution is able to work in a stable manner for a case of up to 15 messages in the buffers of every CB's neighbors.

```
int packet_in_memory (int packet.id, int packet.seq))
   {
   buffer.temp = 0  // borrows the for the moment unused buffer memory
                    // to store bit variable before the function returns
   for (int i = 0; i<48; i++)
      if (callee[i].id == packet.id)
         if (callee[i].seq == packet.seq)
            {
            buffer.temp = 1
            calle[i].db = 1; // The dirty bit of existing entry is set
            }                // to 1 as a precaution for the case that
                             // several versions of the packet are still
                             // waiting to be sent in the queues of
                             // other neighbors.
         else
            {
            callee[i].seq = packet.seq;
            callee[i].db = 1;
            buffer.id = packet.id;
            buffer.temp = packet.temp;
            buffer.seq = packet.seq;
            link_send (buffer, 50);
            }
   return buffer.temp;
   }
```

Appendix D – Estimating the Traffic Reduction for 4.1.

Figure 10 shows a way of how the average traffic reduction of solution 4.1. as compared to 3. might be approximated for a square topology.

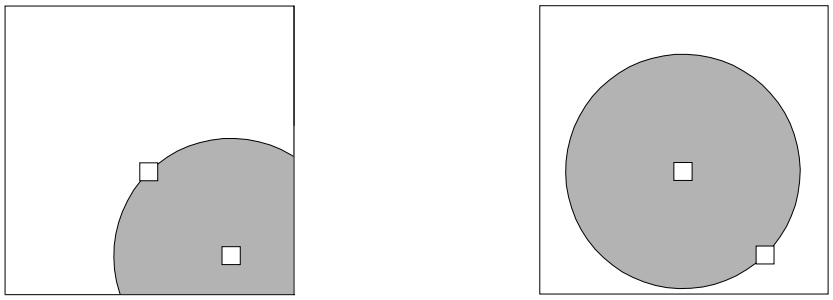


Figure 10: Approximate Traffic Reduction

$$F = \overline{\frac{Square\ Area}{Shaded\ Area}}$$

F, the approximated traffic reduction factor, is the average value for the fraction of square areas to shaded areas in both directions of all possible constellations. Lacking the mathematical tools to calculate this factor, I estimate it to roughly approach the value 4 for $n \to \infty$.

Again, this is an average factor and relying on it would not permit to establish a guaranteed systems response.