Solution of Problem Set 2

October 7, 2005

Question 1(5.13):

(a) If the on-line adversary is able to know the key k at the very beginning before any process sends out messages, he can always fail the *RandomAttack* algorithm. What he needs to do is just to let through all messages before the round k, block all messages to exactly one process i and let through others in the round k, and then block all messages after the round k. Then process i will disagree with other processes. So ϵ is always 1.

If the on-line adversary is unable to know the key k until the end of the first round, then he has to make a guess about his strategy for the first round. If he lets through all the messages in the first round, and the key k happens to be 1, the *RandomAttack* algorithm will succeed. Otherwise, he may block some messages. When $n = 2$ or $r \leq 2$, the adversary can always fail the algorithm by blocking all the messages sent to exactly one process while letting through others in the first round. When $n \geq 3$ and $r \geq 3$, in the round 3 any process's level can at most reach 2, so the maximum level in the network can only reach $r - 1$ at the end of the round r. If the key k happens to be r, the *RandomAttack* algorithm will succeed. From the above, we conclude that when $n \geq 3$ and $r \geq 3$, $\epsilon = \frac{r-1}{r}$.

(b) Thanks to Lev and Nikhil's algorithm which may provide an upper bound to ϵ. I believe it does work, so ϵ is not necessary to be 1 as many believe. The algorithm only takes one round, at the end of which each process i does the following:

1. If $\text{input}_i = 0$, output 0;

2. else, if any 0 message is received, output 0;

3. else, if receive 1 messages from all the other nodes, output 1;

4. else, if some messages are missing, flip a coin to decide output.

It is easy to verify the validity of the algorithm. For any adversary, if he lets through all the messages, the algorithm will succeed; otherwise, there is still a small successful probability of at least $1/2^n$ coming from flipping a coin in each process which is independent of any adversary's effect. Therefore, ϵ is less than $1 - 1/2^n$, so is the lower bound.

Question 2(6.34):

According to **Theorem 6.29**, $n > 3f$ and $conn(G) > 2f$.

(a) The connectivity of a ring is 2, so Byzantine agreement algorithms can not tolerate any faulty process in a ring.

(b) The connectivity of a three-dimentional cube is 3 for the degree of any corner process is 3 and $n \geq 8$. Hence, Byzantine agreement algorithms can tolerate at most 1 faulty process in a three-dimentional cube.

(c) The connectivity of a complete bipartite graph with m nodes in each of its two components is m. Therefore, Byzantine agreement algorithms can tolerate $\lfloor \frac{m-1}{2} \rfloor$.

Question 3(6.45):

(a) Starting with $exec(\rho_0)$, now we want to change process 1 from 0 to 1 without being noticed by any other process, so we must remove all the messages that process 1 sends in every round, i.e. make process 1 stop running at the very beginning. Let $T_k, k = 1, \ldots, f$ indicate the number of executions in order to make process i stop after the end of round $f - k$. Notice that in order to make consecutive executions indistinguishable to some nonfaulty process, we can only add or remove one message each step. For the final round f, i.e. $k = 1$, it is easy to see that we only need to directly remove all the messages process 1 sends one by one, i.e. $T_1 = n - 1$, in which n is the number of process. For $k > 1$, in order to remove the message m process 1 sends to another process $i, i > 1$, we must make process i stop after the end of round $f - k + 1$(which may be called *killing witness*), otherwise, process i can tell others about this missing message. Then recover process i after removing m for the constraint of maximum faulty processes. The

total cost to make process i stop and then restart is $2T_{k-1}$ according to our definition. So, the cost of removing all the messages process 1 sends in the round $f-k+1$ is $(n-1)(2T_{k-1}+1)$. Plus the cost of making process 1 stop after the round $f-k+1$, we can obtain $T_k = (n-1)(2T_{k-1}+1) + T_{k-1}$, indicated as the following:

$$\begin{aligned} T_1 &= n-1 \\ T_2 &= (n-1) + (2n-1)T_1 \\ &\cdots \\ T_k &= (n-1) + (2n-1)T_{k-1} \end{aligned}$$

Finally we can get that

$$T_k = \frac{(2n-1)^k - 1}{2}.$$

So the cost of making process 1 stop at the very beginning is $T_f = \frac{(2n-1)^f - 1}{2}$. Counting that changing the input of process 1 costs one execution and restoring all deleted messages that process 1 should send in nonfaulty condition costs another T_f, the total cost of changing process 1's input should be $2T_f+1$. There are n such processes, so the length of the final chain should be

$$L = n(2T_f + 1) = n(2n-1)^f.$$

Actually, the above calculation is only approximate, and the actually length is smaller when $f \geq 3$. Note that for $k \geq 3$, when we want to remove the message process 1 sends to process i, we must make process i stop after the end of round $f-k+1$. Also, we want this process i's stopping won't be noticed by other processes, so we have to continue to kill other witnesses, as implied in the above inductions. However, process 1 has already stopped after the end of round $f-k+1$, so we don't need to kill it again. Hence, the induction equation should like that

$$T_k = (n-1)+2(n-1)(T_{k-1}-2T_{k-2})+T_{k-1} = (n-1)+(2n-1)T_{k-1}-4(n-1)T_{k-2}.$$

If you are insterested, you can try to calculate the above equation. :D

(b) If we use Byzantine faults, we can let processes directly tell a lie, i.e. a process can send a faulty message with value opposite to its input. Then in order to change a process's input, we don't need to make it stop at the very beginning(recall that in order to do that, we have to delete all its messages

and restore the messages again!), but just revert the values of its messages one by one. For example, if $n = 3$ and $f = 1$, we still start the chain with $exec(\rho_0)$, and want to change process 1's input from 0 to 1. Let process 1 lie to process 2 that its value is 1. This execution is indistinguishable to process 3. Then let process 1 lie to process 3, and this execution is indistinguishable to process 2. Now, we can change process 1's input directly to 1, and process 1 becomes nonfaulty. Finally, we will end up with $exec(\rho_3)$, where all processes have input value 1 and there are no failures. So we have the needed chain, which gives a contradiction.

When $f > 1$, we can get the similar chain and induction equations as (a):

$$\begin{aligned} T_1 &= n-1 \\ T_2 &= n-1+(n-1)T_1+T_1 = n-1+nT_1 \\ &\cdots \\ T_k &= n-1+(n-1)T_{k-1}+T_{k-1} = n-1+nT_{k-1} \end{aligned}$$

So we obtain $T_f = n^f - 1$, and the total length of the chain is $n(T_f+1) = n^{f+1}$. So we reduce the length almost by a factor 2^f.