A Distributed Protocol For Adaptive Link Scheduling in Ad-hoc Networks[1]

Rui Liu, Errol L. Lloyd
Department of Computer and Information Sciences
University of Delaware
Newark, DE 19716

Abstract -- A fully distributed protocol for adaptive link scheduling in multi-hop ad-hoc networks is presented. The distributed and adaptive nature of ad-hoc networks makes centralized algorithms inappropriate for use in practice. The protocol presented here is fully distributed and does not require any global information other than an assumption of time synchronization. The protocol handles incremental changes in the network topology through local actions without involving or affecting stations that are geographically removed from the change. Concurrent topology changes are permitted if they are separated by at least three hops. Experimental results establish that the protocol is competitive with centralized non-adaptive methods both in running time and in the quality of the schedule.

Keywords—Ad-hoc network, link scheduling, distributed protocol, adaptive algorithm

1. Introduction and background

An *ad-hoc network* is a wireless network infrastructure that supports mobile communication. Such networks are fully distributed, quickly deployable, multi-hop systems. Communication channels are shared by all of the stations in an ad-hoc network, thereby enabling the transmission of a station to be received by all stations within its transmission range. To achieve robust and collision free communication, there are two alternatives. One is to utilize a random access MAC layer scheme. The other is to construct a *transmission schedule*. One variant, *link scheduling* in the context of time division multiplexing (TDM) is the subject of this paper.

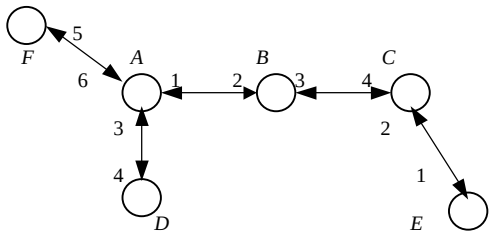
1.1 Link scheduling fundamentals

In link scheduling, it is guaranteed that a scheduled transmission on a link $x \rightarrow y$ will not result in a collision at either x or y. In this context, two types of collisions must be avoided. *Primary interference* occurs when a station transmits and receives at the same time. *Secondary interference* occurs when a station simultaneously receives two or more separate transmissions. Thus, two links assigned to the same time slot *conflict* if they are adjacent to the same station (primary interference) or if there exists a third link from the transmitter of one link to the receiver of the other link (secondary interference). In a link schedule, there may be no conflicts.

Formally, a *link schedule* consists of a sequence of fixed-length time slots, where each link is assigned a *time slot* and a *transmission cycle*. A station may transmit over a link in the time slot assigned to that link once every transmission cycle. Since the transmission of a station is

[1] Prepared through participation in the Advanced Telecommunications-Information Distribution Research Program (ATIRP) Consortium sponsored by the U.S. Army Research Laboratory under Cooperative Agreement DAAL01-96-2-0002.

intended for a particular neighbor (i.e. links are being scheduled), the schedule must be such that there are no collisions at the endpoints of the links scheduled in a particular time slot. Note that transmission cycles may vary from link to link. As in prior works, we assume that the network is a *unit disk network*: links are bi-directional, each station has an identical transmission range R, and that the transmissions of a station will be received by all stations within a Euclidean distance of R. An example of a link schedule appears in Figure 1. There, using the algorithm of section 2.1, link $E \rightarrow C$ has transmission cycle of 4, while all other links have transmission cycles of 8.



Slot	1	2	3	4	5	6
A		B		D	F	
B	A			C		
C	E		B			
D			A			
E		C			C	
F						A

Fig. 1. Example of link scheduling.

1.2 Adaptive link scheduling

Classic applications of ad-hoc networks are battlefield communication, disaster recovery, and search and rescue. In these situations, ad-hoc networks must be rapidly deployable and highly scalable. Unfortunately, all of the existing link scheduling algorithms are centralized off-line methods that require global knowledge of the network[2]. Whenever the topology of the network changes, all of the stations in the network will have to wait until a new schedule is centrally produced. This makes the network highly non-scalable.

An alternative approach is to utilize *adaptive algorithms*, which, given a link schedule for the network, and a change in the network (by the joining or leaving of a station), appropriately updates the schedule to correspond to the modified network. The twin objectives of adaptive algorithms are much faster execution (than an off-line algorithm that computes a complete new schedule) and the production of a *high quality* schedule. Note that the joining of a station will introduce new links between the new station and its one-hop neighbors. These new links must be assigned time slots before they can be utilized.

1.3 An overview of earlier results

A natural goal for link scheduling is to produce a schedule where the maximum transmission cycle is minimized. Unfortunately, producing such optimal schedules is NP-complete [1], even

[2] The one existing distributed algorithm [8] for related problems operates under the strong assumptions that: 1) each station uses a unique quasi-orthogonal CDMA channel, and 2) the total number of stations in the network must be known by every station. As such, the algorithm is not compatible with the problem framework utilized in this paper.

for networks that may be modeled by a planar graph. For that reason, the research focus has been on heuristic algorithms [2,4,7,9,10]. Two notable heuristics (both of which utilize a uniform transmission cycle that is equal to the largest time slot used by any link) are:

- *Pure Greedy* [2]: Links are scheduled one at a time in an arbitrary order, with a link assigned the lowest numbered time slot that does not create any conflict with the links that have already been assigned time slots. Pure greedy is often the method of choice for link scheduling due to its simplicity. It has a worst case running time of $O(\rho^2 e)$ in unit disk networks, where e is the number of edges in the network, and ρ is the maximum degree of any station in the network.
- *DLS* [2]: This algorithm distributes the computation of the schedule (by use of a token), but does not allow parallel computation of the schedule. Its performance is related to the *thickness* of the network (thickness is the minimum number of planar graphs into which the network can be partitioned). This is the only link scheduling algorithm that does not rely upon the existence of a central site for computing the schedule. It does require the use of global information at each station (passed along with the token). It has a worst case running time of $O(n \log n + \theta\rho e)$ in unit disk networks, where n is the number of stations in the network.

2. *FUDIL* - Fully Distributed Link Scheduling

The primary result of this paper is a *fully distributed* link scheduling algorithm (***FUDIL***). Notable features of ***FUDIL*** are:

- No global information is required. This means both that there is no central site, nor do the individual sites require any global information.
- The method is fully distributed, allowing stations that are not geographically close to run the scheduler simultaneously.
- Standard TDM slot synchronization is utilized. It is assumed that this is available through GPS or analogous technology. Since this requires only that each station be capable of reading a GPS signal, no uplink to the GPS satellite is necessary.

The specification of ***FUDIL*** is given in three phases. In section 2.1, we describe the scheduling method as a centralized, nonadaptive algorithm. In section 2.2, we describe how to make the method adaptive. Finally (section 2.3), we indicate how to make the method fully distributed.

2.1 The basic scheduling method of *FUDIL*

In this section we describe the basic scheduling method as a centralized, nonadaptive method. Recall that nonadaptive means that information about all of the network stations is available at the outset to the centralized site. Using this information, each link is first assigned a transmission slot according to the following algorithm.

```
Method Assign_All_Slots()
        Mark every link as unassigned;
        For each station A do
                T_Slot_Assign(A);

Method T_Slot_Assign(A)                 // A is a station
        Let Link_List(A) contain all outgoing links of A;
        For every ℓ in Link_List(A) do
                Conflict_Link_List(ℓ) ← all links that conflict[3] with ℓ;
                Con(ℓ) ← Number of different transmission slots utilized by links in Conflict_Link_List(ℓ);
        Sort Link_List(A) in non-increasing order according to Con(ℓ);
        While Link_List(A) ≠ ∅ do
                Let ℓ be the first link in Link_List(A);
                T_Slot(ℓ) ← Least transmission slot not utilized by links in Conflict_Link_List(ℓ);
                Delete ℓ from Link_List(A);
```

Note that in this transmission slot assignment, the uncolored links around a given station are assigned slots from the most to the least constrained. This allows for the highest likelihood that a low numbered slot will be utilized for each link. The running time of **T_Slot_Assign** is $O(\rho^3)$ in networks modeled by unit disk graphs.

Once each link has been assigned a transmission slot, links are assigned transmission cycles.

```
Method Assign_All_Cycles()
        // Conflict_Link_List(ℓ) information should be available for every link ℓ;
        For every link ℓ do
                Max_C(ℓ) ← max(T_Slot(u): u ∈ {ℓ, Conflict_Link_List(ℓ)});
                T_cycle(ℓ) ← min{2^i: 2^i ≥ Max_C(ℓ)}
```

Clearly, the transmission cycle of a link is equal to the least power of two[4] that is greater than or equal to the largest of the transmission slots utilized by ℓ or utilized by a link that conflicts with ℓ. The order in which links are assigned transmission cycles is irrelevant – the same set of cycles will be assigned regardless of the order. The running time of **Assign_All_Cycles** is $O(e\rho^2)$

[3] Recall from section 1.1 that two links ***conflict*** if either: 1) they are adjacent to the same station (primary interference); or, 2) there exists a third link from the transmitter of one link to the receiver of the other link (secondary interference).

[4] Actually, it follows from the proof of Theorem 1 that any integer i>1 will work in place of 2, provided all stations utilize the same i. Intuitively however, 2 provides the highest level of schedule efficiency, hence it is the value we utilize.

which is $O(n\rho^3)$. It follows that the total time for executing the basic scheduling algorithm is $O(n\rho^3)$ in a unit disk network

That the algorithm produces a functionally correct schedule follows from the next theorem. A proof of this result appears in the Appendix.

Theorem 1 *For any two links A and B that conflict, if A and B are assigned transmission slots and cycles as specified above, then A and B never transmit in the same slot.*

2.2 Towards *FUDIL*: making the basic method adaptive

In this section we describe modifications to the basic scheduling method so that it adaptively handles the joining or leaving of a station.

Joining of Stations: Consider adjusting the schedule when a single station, and its up to 2ρ links, joins the network. Since these new links cannot introduce conflicts between existing links, the schedule can be updated by executing T_Slot_Assign and then re-computing the transmission cycles for each link that is in conflict with one of the newly added links. The worst case running time for the joining of a station is $O(\rho^3)$, which is an order of magnitude (i.e. a factor of n) less than simply rerunning either pure greedy, $O(e\rho^2)$, or our basic scheduling method, $O(n\rho^3)$.

Leaving of Stations: Consider adjusting the schedule when a single station (and all of its links) leave the network. Such a change will never introduce conflicts into the schedule. It may be however that some links will have a transmission slot/cycle that is larger than necessary. Such links may be far removed from the station that leaves the network, due to a potentially long sequence of links whose transmission slot may be reduced, and then whose neighbors may have their slots reduced. We believe that such sequences are rare and we make adjustments in the schedule only for links that are in conflict with deleted links. The method is as follows:

```
Method Delete(A)                                   // A is the station that is leaving
    Conflict_List(A) ← all links that conflict with a link incident on A;
    For every ℓ in Conflict_List(A) do
        Conflict_Link_List(ℓ) ← all links that conflict with ℓ ;
        Con(ℓ) ← Number of different transmission slots utilized by links in Conflict_Link_List(ℓ);
        If T_Slot(ℓ) > Con(ℓ)                      // checks if a change is needed
            Then T_Slot(ℓ) ← Least transmission slot not utilized by links in Conflict_Link_List(ℓ);
    For every ℓ in Conflict_List(A) do
        Max_C(ℓ) ← max(T_Slot(u): u ∈ {ℓ, Conflict_Link_List(ℓ)});
        If T_cycle(ℓ) ≥ Max_C(ℓ)*2                 // checks if a change is needed
            Then T_cycle(ℓ) ← min{2^i: 2^i ≥ Max_C(ℓ)};
```

From a judicious maintenance for every link of it's Conflict_Link_List, as well as information on the transmission slots utilized by links in that list, and the use of dynamic tables, Delete can be implemented in time $O(\rho^4)$ for a unit disk graph.

2.3 *FUDIL*: A fully distributed protocol

Although both the basic scheduling method and the adaptive version are centralized algorithms, neither requires global network knowledge to schedule any individual link. Rather, only information about that link, and links that conflict with that link, is required. This fact enables a fully distributed protocol for link scheduling. Specifically, from the perspective of a single station joining the network, scheduling the links associated with the station is easy, since the station simply needs to execute the relevant adaptive algorithm from the previous section. However, complications arise when changes in close-by portions of the network occur close together with respect to time. In that situation, significant coordination is required to ensure that conflicts do not occur. That coordination is the subject of this section.

• **Distributed Method- Joining of Stations:** The joining process utilizes two steps. The first, *registration*, ensures that one and two-hop neighbors are prevented from joining the network simultaneously. In the second, *resolution*, the actual schedule modifications are performed. To facilitate these steps, we introduce a special time slot (a *J_slot*) into the schedule every k slots, where k is a constant. A J_slot contains 5 sub-slots named **B**, **R**, **C**, **H**, and **A**. The lengths of these 5 sub-slots are not necessarily the same.

Registration: Two sets of actions need to be described: those of the joining station *A*, and those of online stations.

In regard to joining station *A*, at the first J-slot, *A* listens to sub-slot **B**. If *A* detects a transmission in **B**, then *A* knows there is a station within two hops that is in the process of joining the network. In this case, *A* will wait a random number of J-slots and try again. If there are no transmissions in **B**, then *A* will broadcast a ***registration request*** in sub-slot **R**. This transmission is heard (perhaps by recognizing a collision in case of multiple stations making such a transmission) by all one-hop neighbors of *A*. If *A* does not detect any collision in that broadcast, then it listens to sub-slot **C**. If there are no transmissions in **C**, then *A* will claim itself as a ***head*** by broadcasting its ID in the subsequent **H** sub-slot. This transmission will be heard collision free by all of the one-hop neighbors of *A*. If *A* detects a collision in sub-slot **R** or hears anything in sub-slot **C**, then *A* knows that other stations within two hops are attempting to join the network.

Thus, *A* will wait a random number of J-slots and try again (to execute the full Registration procedure).

In regard to an online station *S*, if *S* detects a collision in sub-slot **R**, then *S* will broadcast in sub-slot **C**, otherwise *S* will remain silent. If *S* hears A's registration information in sub-slot **H**, then *S* will record *A* as it's head and broadcast in all following sub-slots **B** until the joining process is complete.

Note that registration will be completed in one J-slot if no stations that are within two hops are simultaneously attempting to join the network.

Resolution: In this step a station that is now a head completes the scheduling of its links and also initiates subsequent changes in the schedules of nearby links. In so doing, the joining station *A* (a head): collects information on the schedule from its one and two hop neighbors; uses that information to schedule its own links; and, transmits its schedule back to its one and two hop neighbors, whereby those neighbors update the transmission cycles (but not slots!) of their links. The collection and transmission of information by station *A* is complicated by two factors:

- The links of *A* are not yet scheduled, so *A* must communicate with its neighbors by other means.

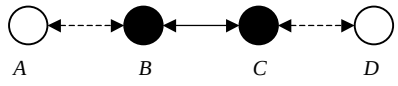


Fig. 2

- Two heads that are three hops away need to coordinate their scheduling so as to avoid conflicts. For example, in Figure 2, suppose A and D are joining stations, and B and C are online stations. The link(A→B) conflicts with link(C→D).

These complications are handled by having *A* communicate with its one hop neighbors via particular subslots of the J-slots, and by having its one and two hop neighbors utilize their already scheduled links to transmit information to and from the two hop neighbors of *A*. The details of this coordination are omitted due to space constraints.

Distributed Method - Leaving of Stations: Due to space constraints we omit the specification of the method for handling the deletion of stations. We note only that the total number of messages sent in processing the deletion of a station is $O(\rho^3)$.

3. Experimental Results

Recall that the alternative to the use of fully distributed adaptive algorithms is to utilize off-line centralized algorithms that re-compute the entire schedule after each change to the network. In that regard, the primary performance issue for fully distributed algorithms is:

What is the quality of the generated schedule relative to a schedule produced off-line?

Recall from section 1 that the two primary off-line algorithms are Pure Greedy and DLS. In this paper the performance of *FUDIL* is compared only against Pure Greedy. There are two reasons. First, the running time of DLS is prohibitive even for networks of moderate size [2]. Second, prior work [2] established that DLS outperformed Pure Greedy in practice by only 4 to 12%. These two facts make Pure Greedy the algorithm of choice for practical purposes.

This section details results for 24 sets of experiments, each of which is generated and maintained on a 400 by 400 grid. The experiments were generated by varying:

1. Station placement within the network – there are two alternatives:

- Stations are placed in the network area using a uniform random distribution.
- The placement of stations in the network area is determined as follows: 20% of the stations are uniformly randomly placed into two 100 by 100 areas which are at two diagonal corners, and the other 80% of the stations are placed in the whole 400 by 400 area using a uniform random distribution (i.e. 30% of the stations are in 12.5% of the area). This placement pattern is referred to as a *skewed placement*.

2. The station generation pattern – there are three alternatives:

- The initial network is empty. Then, 1000 stations join (one at a time). After each insertion, the schedule is updated using ***FUDIL***. Pure Greedy is run on the complete network of 1000 stations.
- The initial network consists of 1500 stations, scheduled using pure greedy. Then, 500 randomly chosen stations are deleted (one at a time). After each deletion, the schedule is updated using ***FUDIL***. Pure Greedy is run on the final network of 1000 stations.
- The initial network consists of 1000 stations, scheduled using pure greedy. Then, 300 changes are made (one at a time). Each change consists of, with equal probability, one station either joining or leaving the network. After each change, the schedule is updated using ***FUDIL***. Pure greedy is run on the final network of 1000 stations.

3. The station transmission range: The station transmission range was varied between 15, 25, 35 and 45 grid units. Within each trial all stations have an identical transmission range.

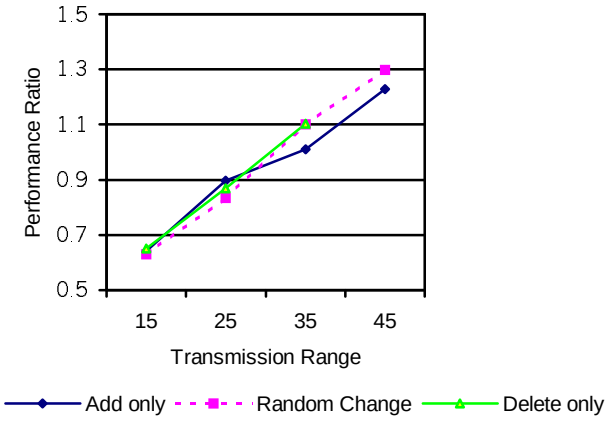
The networks generated as a result of these experiments varied in average maximum one-hop degree from 12 to 56 when the stations are uniformly distributed and from 19 to 110 in the skewed placement. The results of the experiments are shown in Graphs 1 and 2. There, in evaluating the algorithm performance we utilize a ***performance ratio*** obtained by comparing:

> *The average transmission cycle of a station as produced by **FUDIL**, against the average transmission slot of a station as produced by Pure Greedy.*

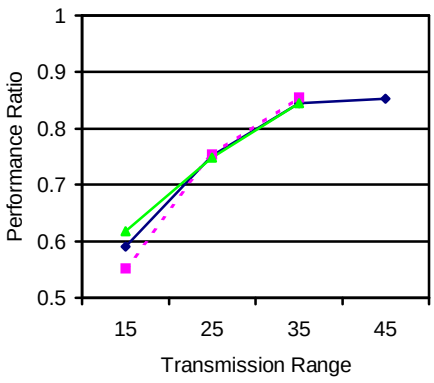
Graph 1 shows the results of each experiment based on a uniform placement of stations, and graph 2 shows the results for the skewed placement. In each graph there are three curves, corresponding to the three station generation patterns. Each curve plots the performance ratio of *FUDIL* for that station generation pattern for the four possible transmission ranges. Each point on those curves represents an average over ten trials using those parameters. From graphs 1 and 2 we make the following observations and conclusions:

- In a sparsely connected network (small transmission range) ***FUDIL*** consistently produces higher quality schedules than does pure greedy. This finding is significant since sparsity is often the norm in ad-hoc networks (and ***FUDIL*** is fully distributed!).
- In the network with skewed placement of stations, ***FUDIL*** has a consistently superior performance to Pure Greedy, even when the transmission range is large (hence the maximum one-hop degree is large).

Graph 1



Graph 2



- The quality the schedule produced by of ***FUDIL*** is quite uniform, in that it does not depend on the type and order of network changes (i.e. joinings and leavings of stations).

Also of interest is the relationship between various network and schedule parameters:

- In a sparse network, the maximum transmission cycle is many times the minimum transmission cycle. When the transmission range is 15, the average transmission cycle is about 40% of the sum of the maximum and minimum transmission cycles. In a crowded network, the maximum transmission cycle is two to four times larger than the minimum transmission cycle. When the transmission range is 45, the average transmission cycle is about 80% of the sum of the maximum and minimum transmission cycles.
- Since Pure Greedy assigns the same transmission cycle to all stations, while *FUDIL* does not, it may seem that *FUDIL* is unfair. We argue that this is not the case, since the reason that some stations can transmit less frequently is that they are in a crowded area, which fundamentally restricts the frequency of transmissions in that area. By contrast, there is no reason to penalize stations in uncrowded areas by insisting that they utilize the same cycle as stations in crowded areas. By allowing for different transmission cycles, *FUDIL* significantly decreases the *average* transmission cycle.

Bibliography

[1] S. Ramanathan and Errol L. Lloyd, "Scheduling algorithms for multi-hop radio networks", IEEE/ACM Transactions on networking, VOL 1, NO.2, April,1993.

[2] Errol L. Lloyd and S. Ramanathan, "Efficient distributed algorithms for channel assignment in multihop radio networks", Journal of High Speed Networks, 2:405-423, 1993.

[3] D. Bersekas and R. Gallager, Data Networks. Englewood Cliffs, NJ: Prentice-Hall, 1987.

[4] I. Chlamtac and S. Kutten, "A Link Allocation Protocol for Mobile Multi-Hop Radio Networks", in Proc. GLOBECOM, 1985.

[5] Xiaopeng Ma and Errol L. Lloyd. "Practical adaptive algorithms for channel assignment in multihop radio networks", ATIRP Conference, 1998.

[6] Xiaopeng Ma and Errol L. Lloyd. A distributed protocol for adaptive broadcast scheduling in packet radio networks

[7] I. Chlamtac and S. Lerner, "A Link Allocation Protocol for Mobile Multi-Hop Radio Networks", in Proc. GLOBECOM, 1985

[8] D. J. Baker, A. Ephremides, and J. A. Flynn, "The Design and Simulation of a Mobile Radio Network with Distributed Control", in IEEE journal on selected areas in cummunications, VOL. SAC-2, NO. 1, Jan 1984

[9]A.Ephremedis, J. E. Wieselthier and D.J. Baker, "A Design Concept for Reliable Mobile Radio Networks with Frequency Hopping Signalling", Proc. IEEE 75(1), (Jan. 1987), 56-73.

[10] R. Ogier, "A decomposition method for optimal scheduling", in Proc. 24th Allerton Conf., Oct 1986.

APPENDIX

The Proof of Theorem 1

Let A and B be arbitrary links that conflict. Suppose that after executing **Assign_All_Slots** and **Assign_All_Cycles** the transmission slot for A (B) is S_A (S_B) and the **Max_C()** value for A (B) in **Assign_All_Cycles** is C_A (C_B). It follows from the specification of **Assign_All_Slots** that:

$$S_A \neq S_B \quad (1)$$

Obviously, $$1 \le S_A \le C_A,\ 1 \le S_B \le C_B \quad (2)$$

Further, because A and B conflict, it follows from **Assign_All_Cycles** that:

$$1 \le S_A \le C_B,\ 1 \le S_B \le C_A \quad (3)$$

From **Assign_All_Cycles**, the transmission cycles T_A and T_B can be written as:

$$T_A = 2^p \ge C_A,\ T_B = 2^q \ge C_B \quad (4)$$

where p, q are the smallest integers that satisfy the inequalities.

From (2), (3) and (4): $$\left|S_A - S_B\right| \le 2^p - 1 \quad (5)$$

Suppose by the way of contradiction that links A and B have a collision in the schedule. Then, there are natural numbers s_1 and t_1 such that $S_A + 2^p s_1 = S_B + 2^q t_1$. Thus:

$$S_A - S_B = 2^q t_1 - 2^p s_1 \quad (6)$$

From (6), there are two cases:

Case 1: $p=q$

Here $S_A - S_B = 2^p(t_1 - s_1)$. Thus, if $t_1 = s_1$, then $S_A = S_B$, which is a contradiction to (1). If $t_1 \neq s_1$, then since $t_1 - s_1$ is a nonzero integer, $|S_A - S_B| \ge 2^p$ which is a contradiction to (5).

Case 2: $p \neq q$

Without loss of generality, assume $q>p$, hence $q = p + r$ where r is a positive integer. Then, $S_A - S_B = 2^p(2^r t_1 - s_1)$. Thus, if $2^r t_1 - s_1 = 0$, then $S_A = S_B$ which is a contradiction to (1). But if $2^r t_1 - s_1 \neq 0$, since $2^r t_1 - s_1$ is an integer, then $|S_A - S_B| \ge 2^p$ which is a contradiction to (5).