Vol.2019-AL-173 No.12
2019/5/11

モバイルビザンチン故障の封じ込めと合意形成

半澤　陽[1,a)]　山内　由紀子[2,b)]

概要：複数のプロセスと通信リンクから構成される分散システム中に, 任意の振る舞いを行うビザンチン故障プロセスが存在する時，正常プロセス間で合意を形成する問題をビザンチン合意問題と呼ぶ．さらに，ビザンチン故障プロセスの集合が継続的に変化する場合の合意形成問題を移動ビザンチン合意問題と呼ぶ．Inoue ら (2018) は，ビザンチン故障を引き起こすビザンチンエージェントの移動を制限し，分散システムの一部に封じ込めるために，メッセージの受信を拒否する通信リンクのブロックを提案した．本研究では，ビザンチンエージェントの移動経路に基づいた通信リンクのブロックでは封じ込めが不可能であることを示し，通信リンクを双方向にブロックする切断を用いればビザンチンエージェントの封じ込めが可能であることを示す．さらに，各操作を行いながら移動ビザンチン合意問題を解く分散アルゴリズムを示す．

キーワード：分散システム，合意問題，モバイルビザンチン故障，封じ込め

Agreement with containment of mobile Byzantine faults

Abstract: We consider a distributed system that consists of a set of processes connected by communication links. The Byzantine agreement problem requires that correct processes reach agreement in the presence of Byzantine faults that make faulty processes perform arbitrarily. When the set of faulty processes continuously changes, the problem is called the mobile Byzantine agreement problem. A Byzantine agent visits a process and makes the process faulty, and Inoue et al. (2018) proposed blocking of a link that enables containment of a Byzantine agent to a restricted part of a distributed system. In this paper, we first consider blocking of links based only on the track of a Byzantine agent and we demonstrate that containment is impossible by blocking. Then, we introduce disconnection of a link that blocks a link in both directions and we demonstrate that blocking of links enables containment of a Byzantine agent. We finally present mobile Byzantine agreement protocols with blocking or disconnection of links.

Keywords: Distributed system, agreement problem, mobile Byzantine fault, and containment

1. はじめに

分散システムとは，通信リンクで相互に接続された多数の計算機 (プロセス) からなるシステムである．各プロセスが通信リンクを介して他のプロセスとメッセージを送受信し，局所的に計算を行うことで，すべてのプロセスが協調して動作する．このため，分散システムはプロセスを頂点，プロセス間の通信リンクを辺とするグラフとして表現される．分散システムにおける最も基本的な課題のひとつが全てのプロセスで合意を形成する**合意問題**である．インターネットに代表される大規模な分散システムでは，プロセスや通信リンクの故障，悪意あるユーザーの介入が不可避であるため，様々な耐故障合意形成手法が提案されている．

プロセスが持つメモリの内容を任意の値に書き換える一時故障に対する故障耐性としては，自己安定性が提案されている [2]．**自己安定プロトコル**とは，任意のシステム状況から開始しても自動的に目的のシステム状況へ収束することを保証する分散アルゴリズムである．

故障したプロセスが任意の振る舞いを行う故障を**ビザンチン故障**と呼ぶ．この故障を想定した合意形成問題を**ビザンチン合意問題**と呼ぶ．ここで，合意形成とは，各プロセスが提案する 0 または 1 の提案値のいずれかに，全てのプ

[1] 九州大学大学院システム情報科学府
Graduate School of Information Science and Electrical Engineering, Kyushu Uniersity
[2] 九州大学大学院システム情報科学研究院
Faculty of Information Science and Electrical Engineering, Kyushu Uniersity
[a)] hanzawa@tcs.inf.kyushu-u.ac.jp
[b)] yamauchi@inf.kyushu-u.ac.jp

ロセスが合意することである．n をプロセスの総数，t を故障プロセスの数とした時，Lamport らは，$n > 3t$ の場合，完全グラフ上で合意を形成する分散アルゴリズムを提案し，$n \leq 3t$ の場合，合意を形成する分散アルゴリズムが存在しないことを示した [5]．Garay は故障プロセスの集合が継続的に変化する**移動ビザンチン故障**の下での合意形成問題を提案した [3]．この論文ではプロセス自身が故障していた事を認識できる**故障感知性**をもち，故障することのないプロセスが少なくとも 1 つ存在するという仮定の下で，$n > 6t$ の場合に**移動ビザンチン合意問題**を解く分散アルゴリズムが示された [3]．Buhrman らは，ビザンチン故障プロセスの集合は故障を引き起こすビザンチンエージェントの移動によって変化するとし，各プロセスが故障感知性をもち，ビザンチンエージェントが送信メッセージと共に移動する故障モデルを提案した．彼らは故障することのないプロセスが少なくとも 1 つ存在するという仮定の下で，$n > 3t$ の場合に完全グラフ上で移動ビザンチン合意問題を解く分散アルゴリズムを提案し，$n \leq 3t$ の場合，合意を形成する分散アルゴリズムが存在しないことを示した [1]．Sasaki らは，各プロセスに故障感知性がない場合に，故障することのないプロセスが少なくとも 1 つ存在する場合に，$n > 6t$ の場合に完全グラフ上で移動ビザンチン合意問題を解く分散アルゴリズムを提案し，$n \leq 6t$ の場合，合意を形成する分散アルゴリズムが存在しないことを示した [6]．

Inoue らは，故障感知性に加え，故障から復帰したプロセスがビザンチンエージェントの移動先が分かる探知可能モデルを導入した．さらに，プロセスが隣接プロセスからのメッセージの受信を拒否する通信リンクの**ブロック**を複数行うことで単一のビザンチンエージェントを分散システムの一部に封じ込めながら，分散システム上に全域木を構成する自己安定プロトコルを提案した [3]．このように，ビザンチンエージェントの移動を分散システムの一部に制限することをビザンチンエージェントの**封じ込め**という．

本研究では，通信リンクのブロック，切断によるビザンチンエージェントの封じ込めと合意形成を考える．ここで，**切断**とは通信リンクを双方向にブロックし，以後復帰させないことを指す．本研究ではまず，完全ネットワークで通信リンクを複数本ブロックしても，ビザンチンエージェントをシステムの一部分に封じ込めができないことを示す．次に，通信リンクを切断することで封じ込めができることを示す．さらに，各操作を行いながら移動ビザンチン合意問題を解くアルゴリズムを提案する．

2. 準備

プロセス集合を Π，通信リンクの集合を E とし，分散システムを無向グラフ $G = (\Pi, E)$ と表す．本稿では完全グラフのみを想定する．全プロセス数を n ($|\Pi| = n$) とする．プロセス p_i と p_j について $\{p_i, p_j\} \in E$ である時，p_i と p_j は**隣接**していると言い，p_i の隣接プロセスの集合を $N_{p_i} = \{p_j \in \Pi : \{p_i, p_j\} \in E\} \cup \{p_i\}$ と表す．隣接プロセスは相互にメッセージの送受信による直接通信を行うことができる．プロセスは固有の識別子を持ち，メッセージの送信時に自身の識別子をメッセージに添付することで，受信プロセスは受信したメッセージがどのプロセスから送信されたのかを判別することができる．$\Pi = \{p_1, p_2, ..., p_n\}$ とし，プロセス p_i の識別子を i とする．

すべてのプロセスは**ラウンド**毎に動作し，各ラウンドでメッセージの**送信**，**受信**，**内部計算**を行う．このモデルを**同期モデル**と呼ぶ．プロセスはラウンドの最初にメッセージを送信し，他のプロセスからメッセージを受信し，受信したメッセージに基づいて共通のアルゴリズムを用いて内部計算を行い，自身が管理する変数の値を更新し，次のラウンドで送信するメッセージを決定する．あるラウンドで送信されたメッセージは同じラウンド内で必ず受信される．各プロセスはいくつかの内部変数を管理している．これらの変数の値がプロセスの状態を表す．

故障したプロセスが任意の振る舞いを行う故障を**ビザンチン故障**と呼ぶ．ビザンチン故障は**故障プロセス**の識別子を変えることはなく，メッセージに添付する識別子も変えることはできないとする．故障プロセスは隣接プロセスそれぞれに異なるメッセージを送信することができる二地点間通信を行うものとする．故障プロセスの集合を F と表す．各プロセスは故障プロセスを知ることはできない．$\Pi \backslash F$ に含まれるプロセスを**正常プロセス**と呼ぶ．

移動ビザンチン故障とは，ビザンチン故障プロセスの集合 F がラウンド毎に変化する故障である (図 1)．本研究では，正常プロセスに**ビザンチンエージェント**が訪れることで，そのプロセスが故障すると考える．ビザンチンエージェントはメッセージの送受信と共に移動し，送信と共に送信プロセスを離れ，受信と共に受信プロセスに移動する．分散システム中に高々 t 個のビザンチンエージェントが存在するとする．

各プロセスは直前のラウンドでビザンチンエージェントが滞在していた場合，自身が故障であった事を認識できる**故障感知性**をもつ．直前のラウンドでビザンチンエージェントがいたプロセスを *cured* **プロセス**と呼ぶ．本研究では *cured* プロセスはそのラウンドでは送信を行わず，受信したメッセージをもとに正しい動作に復帰するとする．

移動ビザンチン合意問題は，ビザンチン故障プロセスの集合が継続的に変化する分散システム上で，各プロセスが初期値 $d_i \in \{0, 1\}$ を与えられた時に，正常プロセスが以下の 4 つの条件を満たしつつ，最終的にある値 $v_i \in \{0, 1\}$ で合意した状態に到達する問題である [4]．

（**決定性**）：すべての正常プロセスは合意値 v_i をいずれ決定する．

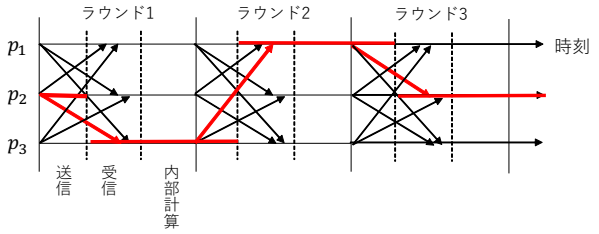

図 1　移動ビザンチン故障の例．各ラウンドの赤い実線はプロセスにビザンチンエージェントが滞在している状態を表し，赤い矢印はビザンチンエージェントの移動を表す．

（**合意性**）：すべての正常プロセスは同じ値で合意しなければならない．
（**妥当性**）：すべての正常プロセスの初期値が同じ値であるならば，その値で合意しなければならない．
（**合意維持性**）：一度正常プロセス間で合意に達すると，その後も正常プロセス間で合意を維持しなければならない．

本研究では故障を起こすことのない常に正常なプロセスが 1 台存在すると仮定する[*1]．

Inoue らは，*cured* 状態のプロセスがビザンチンエージェントがどのプロセスに移動したかを観測できる**探知可能モデル**と観測できない**探知不可能モデル**を提案した [3]．探知可能モデルでは，各プロセスは自身に滞在していたビザンチンエージェントが退出する時に使用した通信リンクとその先のプロセスを，ビザンチンエージェントが退出した次のラウンド，つまり *cured* 状態の時に認識できる．プロセス p_i が故障プロセスの場合を考える．探知可能モデルでは，プロセス p_i は変数 $dest_{p_i}$ を持ち，$dest_{p_i}$ は $\{\perp, 1, ..., n\}$ の要素を値とする．ビザンチンエージェントが p_i から p_j へ移動した次のラウンド，すなわち，p_i が *cured* 状態の時，$dest_{p_i} = j$ となる．探知不可能モデルは，常に $dest_{p_i} = \perp$ の状態である．Inoue らは，探知可能モデルにおいて，隣接プロセスからのメッセージを受信しない通信リンクの**ブロック**を提案した．プロセス p_i がプロセス p_j が送信したメッセージを受信しない時，プロセスは通信リンク $\{p_i, p_j\}$ をブロックしていると言う．

本研究では，隣接プロセス同士が互いを繋ぐ通信リンクを双方向にブロックした状態である通信リンクの**切断**を導入する．切断された通信リンクはその後繋がらないとする．

3. ビザンチンエージェントの封じ込め

グラフ全体を移動することができるビザンチンエージェントが 1 台の場合を想定する．探知可能モデルでは，通信リンクのブロックや切断を行うことによって，ビザンチンエージェントの移動を制限できる可能性がある．ビザンチンエージェントが同じプロセスに常に留まっている状況では，通信リンクのブロックや切断を行うことはできずビザンチンエージェントの封じ込めは行えない．本章では，ビザンチンエージェントが移動し続ける状況を想定し，ビザンチンエージェントの封じ込めを以下のように定義する．

定義 3.1. *(ビザンチンエージェントの封じ込め)*
$X \subsetneq \Pi$ に含まれる全てのプロセスからの通信リンクを $\Pi \backslash X$ の全てのプロセスがブロックしている状態であり，かつビザンチンエージェントが X に含まれているプロセスに滞在している時，ビザンチンエージェントを X に封じ込めたと言う．

ここで，ビザンチンエージェントの移動先のプロセスからの通信リンクのみをブロックする．このようなブロックの手順を**ビザンチンエージェントの移動経路に基づく通信リンクのブロック**と呼ぶ．プロセス p_i がプロセス p_j からの通信リンクをブロックしている時，ビザンチンエージェントはプロセス p_j からプロセス p_i へ移動することができないので，プロセス p_j はプロセス p_i からの通信リンクをブロックすることはできない．このことから，通信リンクをブロックしてもビザンチンエージェントがグラフ全体をハミルトンサイクルに沿って移動する場合，ビザンチンエージェントは全ての頂点を移動し続け (図 2)，ビザンチンエージェントを封じ込められないことがわかる．

定理 3.1. $t = 1$ で各プロセスがビザンチンエージェントの移動経路に基づいて通信リンクをブロックする場合，ビザンチンエージェントを封じ込めることはできない．

証明. ビザンチンエージェントを $X \subsetneq \Pi$ で封じ込めたと仮定する．つまり各プロセス $p_i \in \Pi \backslash X$ から全プロセス $p_j \in X$ へビザンチンエージェントが移動したことになる．X から $\Pi \backslash X$ への通信リンクが存在している時，p_j へ移動したビザンチンエージェントは p_i 以外のプロセスを経由して p_i へ移動する．その経由したプロセスを順に r_1，r_2，…，r_k (r_1，…，$r_m \in X$，r_{m+1}，…，$r_k \in \Pi \backslash X$) とする (図 3)．つまり X に含まれるプロセスが $\Pi \backslash X$ に含まれるプロセスからの通信リンクをブロックしていることになる．通信リンクをブロックしている時，その逆向きの通信リンクはブロックできないのでブロックできない辺が存在することになり，仮定に矛盾する．□

一方，通信リンクを切断することで，ビザンチンエージェントを封じ込めることができる．

定理 3.2. $t = 1$ で各プロセスが通信リンクをビザンチンエージェントの移動経路に基づいて切断する場合，単一のプロセスにビザンチンエージェントを封じ込めることができる．

[*1] 故障しないプロセスが存在しなければ，正常プロセス間で合意することができない [3]

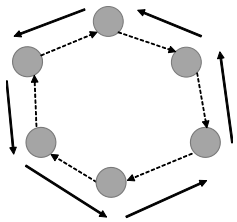

図 2 ビザンチンエージェントの封じ込めが出来ない例．実線矢印はビザンチンエージェントの移動経路，破線矢印はブロックされた通信リンクを表す．

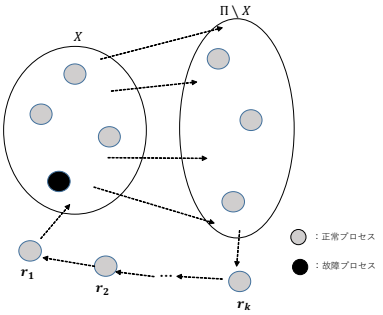

図 3 定理 3.1.4. の証明の参考図．図 3 で破線矢印は通信リンクがブロックされていることを表す．

証明. ビザンチンエージェントが分散システム上を移動し続ける状況では，ビザンチンエージェントが複数回訪問するようなプロセス p_i が存在する．ビザンチンエージェントの移動経路によって通信リンクを切断するので，ビザンチンエージェントが p_i に出入りするために使用される通信リンクは毎回異なる．そのため，最大で $\lfloor (n-1)/2 \rfloor$ 回ビザンチンエージェントがプロセス p_i に入ることで，プロセス p_i は $(n-1)$ 個の隣接プロセスとの通信リンクを全て切断した状況になる．

次に，全プロセス数 n が偶数と奇数の場合について考える．n が偶数で初期状態でプロセス p_i にビザンチンエージェントが滞在していない場合，または n が奇数で初期状態でプロセス p_i にビザンチンエージェントが滞在している場合，プロセス p_i が $(n-1)$ 個の通信リンクを全て切断した時，ビザンチンエージェントは p_i に滞在するので，プロセス p_i にビザンチンエージェントを封じ込めることができる．n が偶数で初期状態でプロセス p_i にビザンチンエージェントが滞在している場合，または n が奇数で初期状態でプロセス p_i にビザンチンエージェントが滞在していない場合，プロセス p_i が $(n-1)$ 個の通信リンクを全て切断した時，プロセス p_i からビザンチンエージェントが他の隣接プロセスへ移動するので，切断後，プロセス p_i にビザンチンエージェントが移動することができなくなり，ビザンチンエージェントが移動できるプロセスが $\Pi \backslash \{p_i\}$ となる．これより，ビザンチンエージェントがグラフを移動し続ける状況では，ビザンチンエージェントは単一のプロセスに封じ込められるか，移動できるプロセス数が 1 個ずつ減少していくかのいずれかである．後者でも，ビザンチンエージェントが移動し続けることで，最終的に単一のプロセスにビザンチンエージェントを封じ込めることができる． □

4. ビザンチン合意アルゴリズム

本節では，完全グラフ上で，ビザンチンエージェントの移動経路に基づいて，通信リンクをブロックした場合と通信リンクを切断した場合の移動ビザンチン合意問題を解くアルゴリズムを与える．

各プロセスは要素がブール値を保持できる配列 *block* を管理し，あるプロセス p_i で，$block_{p_i}[j] = true$ である時，プロセス p_i はプロセス p_j からのメッセージを受信しない．各プロセスの初期値を $d_i \in \{0,1\}$ とする．提案アルゴリズムでは，3 つのラウンドを 1 つのフェーズとし，各フェーズで**コーディネーター**となるプロセスを変更しながら，提案値の変換，更新を行う．正常プロセスがコーディネーターとなる時，正常プロセス間で合意するが，コーディネーターからの通信リンクをブロックまたは切断しているプロセスが存在する場合が考えられる．そのような場合は次のフェーズの第 1 ラウンドに正常プロセス間で合意を形成する．コーディネーターとなるプロセスが $(n-4)$ 個以上のプロセスからブロック，切断されていない時，コーディネーターが正常プロセスであるならば，すべての正常プロセスは値 v_i で合意することができる．$(n-4)$ 個以上のプロセスからブロック，切断される時は，常に正常なプロセスがコーディネーターとなれば正常プロセス間で合意することができる．通信リンクをブロックした場合のアルゴリズムを bMopt を Algorithm 1 に示す．正常プロセスは各プロセスから受信した値から自身の値 v_i を決定するが，第 3 ラウンドで *cured* プロセスは Algorithm 2 に示した RECONSTRUCT を実行することで自身の値を決定する．通信リンクのブロックは探知可能モデルを用いていることから Algorithm 3 に示した BLOCK を実行する．プロセスからメッセージを受信しなかった場合はそのプロセスの値 v_i を $\perp$ とする．

プロトコル bMopt について，以下の定理が成り立つ．

定理 4.1. $n > 3$，$t = 1$ の時，プロトコル *bMopt* は完全グラフ上で移動ビザンチン合意問題を解くアルゴリズムである．

通信リンクを切断する場合の移動ビザンチン合意アルゴリズムを dMopt と呼ぶ．プロトコル bMopt と dMopt の相違点は以下の 2 点である．1 点目は，bMopt で BLOCK の操作を行う時に，dMopt では，通信リンクを双方向ブロックする手続きを行う．2 点目は，第 1 ラウンドで cured 状態のプロセスは第 2 ラウンドでは，t 個以上の値 v_i を

Algorithm 1 Protocol bMopt at process p_i

v_i; プロセス p_i の値
c_i; 各フェーズでのコーディネーターとなるプロセスの識別子
MV ; プロセスが受信した値を格納する一次元配列
C; 受信した 0, 1 に対するカウンター
D; 受信した $\perp, 0, 1$ に対するカウンター
ECHO ; プロセスが受信した値を格納する二次元配列

1: $v_i \leftarrow d_i$
2: **for** phase $s = 1$ to ∞ **do**
3: Round 1
4: send v_i to all processes
5: receive(MV[·])
6: **if** cured **then**
7: $BLOCK$
8: **end if**
9: **for** $j = 0$ to 1 **do**
10: $C[j]$ $= \#\ of\ j's$ in MV
11: **end for**
12: $v_i = \begin{cases} 0 & if\ C[0] \geq n' - t \\ 1 & if\ C[1] \geq n' - t \\ \perp & otherwise \end{cases}$
13: *Round* 2
14: send v_i to all processes
15: receive(MV[·])
16: **if** cured **then**
17: $BLOCK$
18: **end if**
19: **for** $j = \perp$ *to* 1 **do**
20: $D[j]$ $= \#\ of\ j's\ in$ MV
21: **end for**
22: $v_i = \begin{cases} 0 & if\ D[0] > t \\ 1 & if\ D[1] > t \\ \perp & otherwise \end{cases}$
23: *Round* 3
24: send MV[·] to all processes
25: **for** *each* i **do**
26: ECHO[i, ·]=MV[·] received from i
27: **end for**
28: **if** cured **then**
29: $BLOCK$
30: **end if**
31: **if** cured in Round 2 **then**
32: RECONSTRUCT
33: **end if**
34: $c_i \leftarrow s\ mod\ n$
35: **if** $v_i = \perp$ *or* $D[v_i] < n' - t$ **then**
36: $v_i = max(0, v_{c_i})$
37: **end if**
38: **end for**

受信した時，その値を自身の値に採用し，それ以外の時は値を $\perp$ とする．第 1 ラウンドで正常であったプロセスは bMopt と同様の手続きを行う．

プロトコル dMopt について，以下の定理が成り立つ．

定理 4.2. $n > 3$，$t = 1$ の時，プロトコル *dMopt* は完全グラフ上で移動ビザンチン合意問題を解き，ビザンチンエージェントが移動し続ける場合，ビザンチンエージェントを封じ込めるアルゴリズムである．

Algorithm 2 RECONSTRUCT

1: **for** *all* $j \in \Pi$ **do**
2: **if** $w \in \{0, 1\}$ occurs at least $n' - t$ times in column j of ECHO[i, ·] **then**
3: $SV_i[j] \leftarrow w$
4: **else**
5: $SV_i[j] \leftarrow \perp$
6: **end if**
7: **end for**
8: **for** $j = \perp$ to 1 **do**
9: $D[j]$ $= \#$ of $j's$ in SV
10: **end for**
11: $v_i = \begin{cases} 0 & (D[0] > t) \\ 1 & (D[1] > t) \\ \perp & (otherwise) \end{cases}$

Algorithm 3 BLOCK

1: **if** $dest_{p_i} > 0$ **then**
2: $block_{p_i}[dest_{p_i}] \leftarrow true$
3: **end if**

定理 4.3. プロトコル *dMopt* は高々 $3n$ ラウンドで，正常プロセス間で合意を形成し，以後合意値を維持する．

定理 4.3 より，合意形成に要する時間はビザンチンエージェントの移動や故障封じ込めに依存しないことがわかる．

5. まとめ

本研究では，1 台のビザンチンエージェントを封じ込める手法と $n > 3$ で移動ビザンチン合意問題を解くアルゴリズムを提案した．今後の課題は，$t > 1$ の場合のビザンチンエージェントの封じ込めと移動ビザンチン合意問題を解くアルゴリズムの考案である．

参考文献

[1] H. Burhman, J. Garay, and J. Hoepman. Optimal Resiliency against Mobile Faults. Proc. of FTCS 1995, pp. 83–88, 1995.
[2] E. W. Dijkstra. Self-stabilizing systems in spite of distributed control. Communications of the ACM, Vol. 17, pp. 643–644, 1974.
[3] J. Garay. Reaching (and Maintaining) Agreement in the Presence of Mobile Faults. Proc. of WDAG 1994, pp. 253–264, 1994.
[4] K. Inoue, H. Kakugawa, and T. Masuzawa. Strongly stabilizing protocol for spanning tree construction with mobile Byzantine. 第 14 回情報科学ワークショップ予稿集, pp. 258–264, 2018.
[5] L. Lamport, R. Shostak, and M. Pease. The Byzantine Generals Problem. ACM Transactions on Programming Languages and Systems, Vol. 4, pp. 382–402, 1982.
[6] T. Sasaki, Y. Yamauchi, S. Kijima, and M. Yamashita. Mobile Byzantine Agreement on Arbitrary Network. Proc. of OPODIS 2013, pp. 236–250, 2013.