Capacity of Byzantine Agreement: Complete Characterization of Four-Node Networks *

Guanfeng Liang and Nitin Vaidya
Department of Electrical and Computer Engineering, and
Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
gliang2@illinois.edu, nhv@illinois.edu

Technical Report

April 6, 2010†

*This research is supported in part by Army Research Office grant W-911-NF-0710287. Any opinions, findings, and conclusions or recommendations expressed here are those of the authors and do not necessarily reflect the views of the funding agencies or the U.S. government.

†A part of this report is submitted for conference review, on February 17, 2010.

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
06 APR 2010		**00-00-2010 to 00-00-2010**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Capacity of Byzantine Agreement: Complete Characterization of Four-Node Networks	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **University of Illinois at Urbana-Champaign,Department of Electrical and Computer Engineering,Coordinated Science Laboratory,Urbana,IL,61801**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**Same as Report (SAR)**	**31**	

1 Introduction

In this paper, we consider the problem of maximizing the throughput of Byzantine agreement. Byzantine agreement is a classical problem in distributed computing, with initial solutions presented in the seminal work of Pease, Shostak and Lamport [22, 15]. Many variations on the Byzantine *agreement* problem have been introduced in the past, with some of the variations also called *consensus*. We will use the following definition of Byzantine agreement: Consider a network with one node designated as the *sender* or *source* (S), and the other nodes designated as the *peers*. The goal of Byzantine agreement is for all the fault-free nodes to "agree on" the value being sent by the sender, despite the possibility that some of the nodes may be faulty. In particular, the following conditions must be satisfied:

- **Agreement:** All fault-free peers must agree on an identical value.
- **Validity:** If the sender is fault-free, then the agreed value must be identical to the sender's value.
- **Termination:** Agreement between fault-free peers is eventually achieved.

Our goal in this work is to design algorithms that can achieve optimal *throughput* of agreement. When defining throughput, the "value" referred in the above definition of agreement is viewed as an infinite sequence of bits. At each peer, we view the agreed information as being represented in an array of infinite length. Initially, none of the bits in this array at a peer have been agreed upon. As time progresses, the array is filled in with agreed bits. In principle, the array may not necessarily be filled sequentially. For instance, a peer may agree on bit number 3 before it is able to agree on bit number 2. Once a peer agrees on any bit, that agreed bit cannot be changed.

We assume that an agreement algorithm begins execution at time 0. The system is assumed to be synchronous. In a given execution of an agreement algorithm, suppose that by time t all the fault-free peers have agreed upon bits 0 through $b(t) - 1$, and at least one fault-free peer has not yet agreed on bit number $b(t)$. Then, the agreement *throughput* is defined as[1] $\lim_{t\to\infty} \frac{b(t)}{t}$.

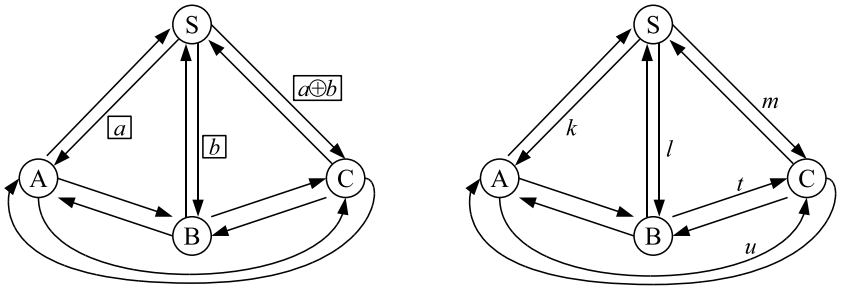
Capacity of agreement in a given network, for a given sender and a given set of peers, is defined as the supremum of all achievable agreement throughputs.

Compared with the traditional Byzantine agreement algorithms, our algorithms can improve the agreement throughput significantly. Consider the network in Figure 1(a), each directed link has capacity equal to R bits/unit time. With the traditional agreement algorithms such as the algorithm by Pease, Shostak and Lamport, the agreement throughput is upper bounded by the minimum capacity of the outgoing links from node S, which in this case is R bits/unit time. On the other hand, our algorithms can achieve agreement throughput arbitrarily close to $2R$, which results in a 100% improvement.

2 Related Work

Prior work on agreement or consensus: There has been significant research on agreement in presence of *Byzantine* or *crash* failures, theory (e.g., [15, 18, 1]) and practice (e.g., [4, 3]) both. Perhaps closest to our context is the work on *continuous consensus* [21, 6, 19] and *multi-Paxos* [14, 4] that considers agreement on a long sequence of values. For our analysis of throughput as well, we will consider such a long sequence of values. In fact, [4] presents measurements of *throughput* achieved in a system that uses multi-Paxos. However, to the best of our knowledge, the past work on multi-Paxos and continuous consensus has not addressed the problem of optimizing throughput of agreement while considering the

[1] As a technicality, we assume that the limit exists. The definitions here can be modified suitably to take into account the possibility that the limit may not exist.



(a) S sends packets $a, b, a \oplus b$ to A, B and C, respectively.

(b) Four node network: Labels denote some link capacities.

Figure 1: Examples of four node networks

capacities of the network links. Some of the past work has analyzed *number of bits* needed to achieve agreement. While this is related to the notion of capacity or throughput, such prior work disregards the capacity of the links over which the data is being carried. Link capacity constraints intimately affect capacity of agreement. Past work has explored the use of error-correcting codes for asynchronous consensus (e.g., [8]). Our algorithms also use error detecting codes, but somewhat differently.

Prior work on multicast using network coding: While the early work on fault tolerance typically relied on replication [5]or source coding [24] as mechanisms for tolerating packet tampering, *network coding* has been recently used with significant success as a mechanism for tolerating attacks or failures. In traditional routing protocols, a node serving as a router, simply forwards packets on their way to a destination. With network coding, a node may "mix" (or *code*) packets from different neighbors [16], and forward the coded packets. This approach has been demonstrated to improve throughput, being of particular benefit in *multicast* scenarios [16, 12, 7]. The problem of *multicast* is related to *agreement*. There has been much research on multicast with network coding in presence of a Byzantine attacker (e.g., [2, 9, 10, 26, 11, 13]). The significant difference between Byzantine agreement and multicasting is that the multicast problem formulation assumes that the source of the data is always fault-free.

3 Summary of Our Previous Works

In our previous technical report [25], we proposed a Byzantine agreement algorithm that achieves arbitrarily close to the supremum of achievable throughputs characterized by a set of sufficient conditions, for the four-node networks. Thus, the supremum of the achievable throughputs specified by these sufficient conditions serves as a lower bound of the agreement capacity. Under some stronger conditions, this algorithm can be made *oblivious* (the notion of oblivious will be elaborated soon in Section 8). A variation of this algorithm is sketched in Section 12.

Our latest technical report [17] provides a complete agreement capacity characterization for the complete four-node network. It introduces the same necessary conditions and tight sufficient conditions as we will be presenting in this paper. We proposed a Byzantine agreement algorithm that achieves capacity. This algorithm is computationally very simple, but is complicated to describe and prove correct. Thus, in this paper, we will first present a capacity achieving algorithm in Section 8 that is computationally somewhat complex, but easier to describe and prove correct. Then in Section 11, we summarize the algorithm we proposed in [17], which is computationally much simpler.

4 Necessary Conditions for Agreement Throughput R

In this paper, we only consider the case when at most 1 node in a network may suffer Byzantine failure. The network is viewed as a directed graph, formed by directed links between the nodes in the network. The following two conditions are *necessary* for achieving agreement throughput of R bits/unit time, while allowing for a single Byzantine failure:

Necessary condition NC1: If any one of the peers is removed from the network, the min-cut from the source S to each remaining peer must be at least R. The proof of necessity of this condition follows by observing that, even if a faulty peer simply stops communicating with the other nodes, the remaining peers still need be able to receive data from the source at rate R.

Necessary condition NC2: The max-flow to each of the peers from the other peers, with the source removed from the network, must be at least R. To argue the necessity of this condition, consider the following two scenarios, wherein node A is one of the peers.
Scenario 1: Assume that peer node A is faulty. So the other nodes are fault-free. Node A misbehaves by pretending that links SA and AS are broken (if these links exist). Thus, node A will not transmit any packets on link AS, and simply discard packets received on link SA. Aside from this, node A behaves correctly. Since S is fault-free, nodes B and C must agree on the information sent by node S.
Scenario 2: Now consider a scenario in which node S is faulty but the other nodes are fault-free. Node S misbehaves only by pretending that links SA and AS are broken (if these links exist).
In essence, both scenarios are equivalent to the scenario in which links SA and AS are broken, but all the nodes are fault-free. Thus, from the perspective of other peers (except node A) these two scenarios are indistinguishable, and they must agree on identical information. In scenario 2, node A is, in fact, fault-free. So it must also agree on the information agreed on by the other peers. The only way for node A to learn this information is to receive it from the other peers (since, in scenario 2, node S has cut off links AS and SA). Thus, the agreement throughput is upper bounded by the min-cut (or max-flow) to A from the other peers, with node S removed from the network.

It turns out that the above conditions remain necessary even when *message authentication* is available. This paper only addresses the case when such authentication mechanisms are **not available**.

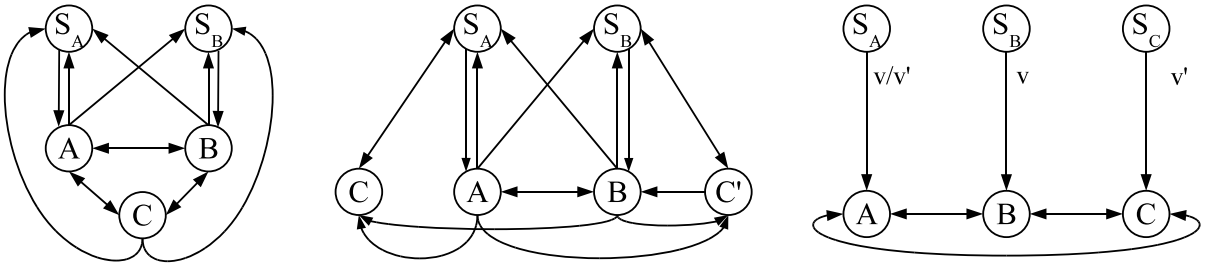
5 Four-Node Networks

It is known that a network must contain at least 4 nodes for agreement to be achievable with a single Byzantine failure. In this section, we explore tight sufficient conditions for achieving agreement at rate R in 4-node networks with at least one feedback link to source. The problem of identifying tight sufficient conditions for arbitrary (larger) networks presently remains open. However, the necessary conditions presented above are applicable to arbitrary networks.

We consider a synchronous network of four nodes named S, A, B and C, with node S acting as the sender, and nodes A, B and C being the peers. At most one of these four nodes may be faulty. Figure 1(b) shows such a four-node network. The labels near the various links denote the link capacities. In this paper, we will refer the outgoing links at the source node S as the *downlinks*, and the incoming links at the source node S as the *uplinks*.

5.1 Necessity of Incoming Links at Peers

In any four-node network, the following condition is *necessary* to achieve a non-zero throughput of agreement.



(a) No link SC. (b) No link CA (c) No uplink

Figure 2: State machines to show necessity of NC3 and NC4. In the first two state machines, S_A wants to send v_A, and S_B wants to send $v_B \neq v_A$.

Necessary condition NC3: All incoming links at the peers must exist.

Proof: We prove this by contradictions. Assume that agreement can be achieve with some incoming links at the peers do not exist. There are two possibilities:
Downlink from S does not exist: Without loss of generality, assume that link SC does not exist. We construct the state machine as shown in Figure 2(a). S_A and S_B run the exact same code as node S in the four node network. Nodes A, B and C run the same code as nodes A, B and C in the four node network, respectively. S_A and S_B are initialized with the values v_A and v_B as the values to send. The uplinks from A and B to the "sources" are broadcast links. In particular, the messages A sends S_A and S_B are identical; the messages B sends to S_A and S_B are identica; and the messages C sends to S_A and S_B are identical. Consider the following two scenarios:

- Scenario 1: S is fault-free and wants to send v_A. B is faulty and behaves in the same way as B in the state machine. In this case, S behaves in the same way as S_A in the state machine; and C behaves in the same way as C in the sate machine, and must decide in v_A.
- Scenario 2: S is fault-free and wants to send v_B. A is faulty and behaves in the same way as A in the state machine. In this case, S behaves in the same way as S_B in the state machine; and C behave in the same way as C in the state machine, and must decide in v_B.

Now we can see that in the state machine, C must decide on both v_A and v_B simultaneously, which leads to a contradiction since $v_A \neq v_B$.
Link from a peer does not exist: Without loss of generality, assume that link CA does not exist. For this part of proof, we construct the state machine in Figure 2(b) in a similar way as the previous one. In this state machine, nodes C and C' run the same code as node C in the four node network. The links from A and B to C and C' are broadcast links. Consider the following two scenarios:

- Scenario 1: S is fault-free and want to send v_A. B is faulty and behaves in the same way as B in the state machine. In this case, S behaves in the same way as S_A in the state machine. Also, A and C behave in the same way as A and C in the sate machine, respectively, and must both decide in v_A.
- Scenario 2: S is faulty. To A, S behaves like S_A in the state machine. To B and C, S behaves like S_B in the state machine. In this case, A, B and C behave in the same way as A, B and C' in the state machine, respectively. To B and C, it is indistinguishable whether S or A is faulty. So they must decide on v_B. Thus, A, which is in fact fault-free, must agree with B and C, and decide on v_B.

Now we can see that in the state machine, A must decide on both v_A and v_B simultaneously, which leads to a contradiction since $v_A \neq v_B$. □

Now we have shown that at least 9 links (3 downlinks from S, and 6 links between the peers) must exist in general four-node networks in order to achieve agreement even for one bit. In the rest of this paper, when we refer to a four-node network, we always assume that NC3 is satisfied.

5.2 When no Uplink Exists

We prove the following necessary condition to achieve agreement throughput of R bits/unit time in a four-node network, when there is no uplink.

Necessary condition NC4: The capacity of every downlink from S must be at least R, when there is no uplink.

Proof: Suppose in contrary, agreement can be achieved at rate $R > l$, with l denoting the capacity of link SA. Thus, for any execution of an agreement algorithm, there must be a time t such that $b(t) > lt$. We will prove this is impossible by proving that agreement on $L > lt$ bits is impossible for any t. Notice that since there is no uplink to the source, the messages S sends to the peers are a deterministic function[2] of the initial value of the data S wants to send (if S is fault free). Then we can construct the state machine shown in Figure 2(c). In this state machine, since $L > lt$, there always exist two different data vectors $v \neq v'$ of length L such that A receives identical messages from S_A. Now consider the following two scenarios:

- Scenario 1: S is faulty. To A, S behaves as S_A in the state machine sending vector v. To B and C, S behaves as S_B and S_C, respectively. For A, this is indistinguishable with the case when S is fault-free and sending v, and C is faulty. So A must decide on v.
- Scenario 2: S is faulty. To A, S behaves as S_A in the state machine sending vector v'. To B and C, S behaves as S_B and S_C, respectively. For A, this is indistinguishable with the case when S is fault-free and sending v', and B is faulty. So A must decide on v'.

This implies that in the state machine, node A must decide on both v and v', which is impossible. The same argument can be applied to any link from S and then we complete the proof.

□

6 Summary of Results

We have two main results.

(1) Four-node network with uplink(s): Agreement **capacity** of a four-node network, with at least one incoming link at S, is the supremum over all throughputs R that satisfy necessary conditions NC1, NC2 and NC3.

(2) Four-node network with no uplink: Agreement **capacity** of a four-node network, with no incoming link at S, is the supremum over all throughputs R that satisfy necessary conditions NC1, NC2, NC3 and NC4.

[2] In this paper, we consider only deterministic algorithms.

7 Byzantine Agreement in Complete Four-Node Networks

It is known that a network must contain at least 4 nodes for agreement to be achievable with a single Byzantine failure. In this section, we explore tight sufficient conditions for achieving agreement at rate R in a *complete* four-node network. The problem of identifying tight sufficient conditions for arbitrary (larger) networks presently remains open. However, the necessary conditions presented above are applicable to arbitrary networks.

We consider a synchronous network of four nodes named S, A, B and C, with node S acting as the sender, and nodes A, B and C being the peers. At most one of these four nodes may be faulty. We assume that the network is *complete*; thus, each pair of nodes is connected by a pair of directed point-to-point links, with the capacity of each link being **non-zero** (although capacity of some links may be arbitrarily close to 0). Figure 1(b) shows such a four-node network. The labels near the various links denote the link capacities. The following two conditions are together sufficient to achieve throughput arbitrarily close to R in a complete four-node network, in presence of a single Byzantine failure.

- **Sufficient condition SC1:** If any one peer is removed from the network, the min-cut from the source to each remaining peer is $> R$.
- **Sufficient condition SC2:** The max-flow to each of the peers from the remaining two peers, with the source removed from the network, is $> R$.

With the notation in Figure 1(b), condition SC1 implies, for instance, that $l + m > R$ and $t + m > R$, and SC2 implies that $t+u > R$. We will show the sufficiency of SC1 and SC2 by presenting a Byzantine agreement algorithm in this section.

Above necessary and sufficient conditions together imply that agreement **capacity** of a complete four-node network is the supremum over all throughputs R that satisfy necessary conditions NC1, NC2 and NC3.

Byzantine agreement algorithms may be *oblivious* or *non-oblivious*: an oblivious algorithm uses a *schedule* of transmissions that is independent of the observed history, whereas a non-oblivious algorithm may adapt the schedule dynamically. In general, to achieve optimal throughput, the agreement algorithm needs to be non-oblivious. However, under certain conditions, an oblivious algorithm can also achieve optimal throughput, as illustrated in Section 12.

The proposed *non-oblivious* Byzantine Agreement algorithm for Complete four-node networks (BAC) has four modes of operation, numbered I, II, III, IV. As seen later, repeated (and pipelined) execution of our algorithm can be used to achieve throughput approaching the capacity. At time 0, the network starts in mode I. The mode number may change as the algorithm is executed repeatedly over time, but it never decreases.[3]

In each mode, the information sent by S is coded and scheduled such that misbehavior by any one node (possibly the sender) is either detected, or the fault-free peers correctly agree without detecting the misbehavior. In the event that a misbehavior is detected, the nodes perform an "extended" round (that is, additional messages) to attempt to narrow down the failure to a subset of nodes. After this *extended* round, each fault-free peer is able to narrow down the failure to a subset that contains one or two of the other nodes. The union of these subsets at the fault-free peers satisfies one of the following three properties: (i) the union contains two peers, or (ii) the union contains one peer and the source node S, or (iii) the union contains just one node. In these three cases, the fault-free peers change the mode of operation to, respectively, modes II, III and IV. The selective use of the *extended* round makes the algorithm *non-oblivious*.

[3]If the algorithm is modified to allow for the possibility of node repair, the mode number may indeed decrease. We ignore node repair in this paper. So at most one node fails for the entire time duration.

7.1 Operation in Mode I

The proposed non-oblivious Byzantine agreement algorithm proceeds in rounds, and achieves throughput arbitrarily close to R, provided that conditions SC1 and SC2 are true. In this section, the units for rate R and the various link capacities are assumed to be *bits/time unit*, for a convenient choice of the *time unit*. We assume that by a suitable choice of the *time unit*, the number R and the various link capacities (such as k and l in Figure 1(b)) can be turned into integers. The algorithm presented in this section is computationally somewhat complex, but easier to describe and prove correct. In Section 11, we will summarize a variation on this algorithm that is computationally much simpler. The algorithm in this section is motivated by Reed-Solomon codes and the prior work on network coding [9]. In particular, for a suitable value of parameter c (as elaborated below), each "packet" sent by the algorithm consists of 1 symbol from Galois field GF(2^c). One execution of the algorithm, which takes multiple rounds, allows the nodes to agree on R symbols from GF(2^c) that are sent by S.

The algorithm executes in multiple rounds, with the duration of each round being approximately c time units (as elaborated in Section 7.5). Note that in c time units, a link with capacity z bits/time unit can carry z symbols (or packets) from GF(2^c). Computation is assumed to require 0 time.[4] As an exception, an "extended round" requires longer duration, but as we will elaborate later, such extended rounds occur at most twice over infinite time interval. Now we present the different rounds for mode I.

Round 1: Node S encodes R packets of data, each packet being a symbol from GF(2^c), into $k+l+m$ packets, each of which is obtained as a linear combination of the R packets of data. Let us denote the R data packets as the data vector

$$\tilde{x} \ = \ [\, x_1, \, x_2, \, \ldots \, , \, x_R \,]$$

and the $k + l + m$ encoded symbols as

$$y_1, \, y_2, \, \cdots, \, y_{k+l+m}$$

For the correctness of the algorithm, these $k + l + m$ symbols (or packets) need to be computed such that any subset of R encoded symbols constitutes **independent** linear combinations of the R data symbols. As we know from the design of Reed-Solomon codes, if c is chosen large enough, this linear independence requirement can be satisfied. The weights or coefficients used to compute the linear combinations is part of the algorithm specification, and is assumed to be correctly known to all nodes a priori. Due to the above independence property, **any** R of the $k + l + m$ symbols – if they are not tampered – can be used to (uniquely) solve for the R data symbols.

In round 1, node S transmits k symbols $y_1, \cdots, y_k$ to node A, l symbols $y_{k+1}, \cdots, y_{k+l}$ to B, and $y_{k+l+1}, \cdots, y_{k+l+m}$ to node C, on links SA, SB and SC, respectively. Other nodes do not transmit. (Please see notation in Figure 1(b).)

Round 2: For packet (or symbol) y_j, we refer to j as its *index*. From the packets received in round 1 from node S, each peer node transmits as many distinct packets as possible to the other two peers, with the packets being selected in increasing order of their index. For instance, the number of packets transmitted by node B to node C is $minimum(l, t)$ (please refer to Figure 1(b) for the notation for link capacities). Thus, node B sends the $min(l, t)$ packets with the smallest index to C, that is, $y_{k+1}, \cdots, y_{k+min(l,t)}$.

[4] As seen later, we use the agreement algorithm in a pipelined manner. Computation delay can be incorporated by adding pipeline stages for computation, in addition to communication stages.

Round 3: Each fault-free peer finds the solution for **each** subset of R packets from among the packets received from the other three nodes in rounds 1 and 2. If the solutions to the various subsets are not unique, then the fault-free peer has detected faulty behavior by some node in the network. In round 3, each peer broadcasts to the remaining 3 nodes a 1-bit notification in indicating whether it has detected a failure or not – to ensure reliability of this broadcast of 1-bit indicators by the three peers a traditional Byzantine agreement algorithm, in particular the algorithm by Pease, Shostak and Lamport [15], is used. Since at most one node is faulty, using this traditional algorithm, all fault-free nodes obtain identical 1-bit notifications from all the peers. If none of the notifications indicate a detected failure, then each fault-free peer agrees on the unique solution obtained above, and the execution of the current instance of the algorithm is complete. However, if failure detection is indicated by any peer, then an "extended round 3" is added to the execution, as elaborated soon. After this, node S replies to all peers with a 1-bit notification indicating whether it will enter the extended round 3 or not, using the same traditional agreement algorithm.

Theorem 1 *In mode I, misbehavior by a faulty node will either be detected by at least one fault-free peer, or all the fault-free peers will reach agreement correctly.*

Proof: Node S is said to misbehave only if it sends packets to A, B and C that are inconsistent – that is, all of them are not appropriate linear combinations of an identical data vector $\tilde{x}$. A peer node is said to misbehave if it forwards tampered (or incorrect) packets to the other peers.

Faulty peer: First, let us consider the case when a peer node is faulty. Without loss of generality, suppose that peer A is faulty. Due to condition SC1, and the manner in which packets are forwarded in rounds 1 and 2, each fault-free peer (that is, B or C) receives R untampered packets, either directly from S or via the other fault-free peer. The solution of these R packets must be the correct R symbols from node S. Thus, any tampered packets sent by node A to B or C cannot cause the fault-free peer to agree on any data other than the correct data $\tilde{x}$. It then follows that, either the fault-free peers agree on the correct data, or detect the faulty behavior by node A (although they will not yet be able to determine that the faulty node is specifically node A) .

Faulty sender S: Now, let us consider the case when node S is faulty. Thus, all the peers are fault-free. For convenience, with an abuse of notation, we will denote the capacity of link XY as XY (instead of using the notation in Figure 1(b)). From condition SC2, it follows that $BA + CA > R$, $AB + CB > R$ and $AC + BC > R$. This implies that, $(AB + BA) + (BC + CB) + (AC + CA) > 3R$. Therefore, at least one of the terms $(AB + BA)$, $(BC + CB)$, and $(AC + CA)$ must exceed R. Without loss of generality, suppose that

$$AB + BA \quad > \quad R \tag{1}$$

Now, let us consider the number of packets nodes A and B exchange in round 2 on links AB and BA together. Observe that A sends $min\{SA, AB\}$ packets to B on AB, and B sends $min\{SB, BA\}$ to A on link BA. So the number of packets they exchange on links AB and BA together is

$$\begin{aligned} min\{SA, AB\} + min\{SB, BA\} &= min\{SA + SB, SA + BA, AB + SB, AB + BA\} &(2)\\ &\geq min\{R, AB + BA\} &(3)\end{aligned}$$

The reason for the $\geq$ in Equation 3 is that each of the first three terms on the right hand side of Equation 2, namely $SA + SB$, $SA + BA$, $AB + SB$, exceeds R, as per condition SC1. Equations 1 and 3 together imply that after round 2, nodes A and B share at least R packets. That is, among

the packets A and B have received, there are R identical packets. Thus, A and B will not agree on different data symbols (since the agreed data must satisfy linear equations corresponding to all received packets), even though S may be misbehaving.

Thus, either at least one of A and B will detect misbehavior by node S, or all the packets they have received will be consistent with an identical data vector (of R symbols). In the former case, the misbehavior by S is already detected (although the fault-free peers may not yet know that S is the faulty node). In the latter case, neither A nor B detect the misbehavior. In this case, we will now consider what happens at node C. In particular, there are three possibilities:

- $AC + CA > R$: In this case, similar to the above argument for nodes A and B, we can argue that nodes A and C will have at least R packets in common, and therefore, they will not agree on two different data vectors. This, combined with the fact that A and B will also not agree on two different data vectors, implies that if none of the three fault-free peers detects a misbehavior, then they will agree on an identical data vector.
- $BC + CB > R$: This case is similar to the previous case.
- $AC + CA < R$ and $BC + CB < R$: In this case, similar to Equation 3, we can show that:

$$min\{SA, AC\} + min\{SC, CA\} \geq min\{R, AC + CA\} \quad (4)$$

$$min\{SB, BC\} + min\{SC, CB\} \geq min\{R, BC + CB\}. \quad (5)$$

  This implies that these four links are all "saturated" (that is, the number of packets sent on each of these links in round 2 is equal to the link capacity). Since $BC + AC > R$ (by condition SC2), it follows that $AC + CA + BC + CB > R$, and that node C has at least R packets in common with the union of packets available to nodes A and B. Since nodes A and B have not detected a misbehavior, these R packets must all be consistent with the solution obtained at nodes A and B both. Thus, node C cannot possibly agree on a data vector that is different from that agreed upon by A and B. Thus, it follows that, either at least one of the peers will detect the misbehavior by node S, or they will all agree.

Extended Round 3: Notice that, if a peer detects an attack, but node S replies that it is not going to enter the extended round, all peers are sure that S is faulty. Then the system enters mode IV.

When S replies with an confirmation, an extended round is added subsequent to a failure detection. We also refer to this extended round as the "broadcast phase". As seen below, the broadcast phase is quite expensive, but it is performed at most twice over time interval $[0, \infty)$. In round 3, the fault-free peers learn that some node has behaved incorrectly. The purpose of the *extended* round is to allow the nodes to narrow down the failure to a subset of the nodes. For this purpose, during the broadcast phase, every node (including the source) broadcasts all the packets it has sent to other nodes, or received from other nodes, during rounds 1 and 2 – as with the failure notifications in round 3, these broadcasts are also performed using the traditional Byzantine agreement algorithm. Since all links in our network have non-zero capacity, it is possible to use the traditional Byzantine agreement algorithm as desired here. However, since the capacity of some of the links may be very small, the time required for performing the *broadcast phase* in extended round 3 may be very large compared to the time required for performing the other rounds.

The fault-free nodes use the information received during the broadcast phase to narrow down the failure to a subset of nodes (as explained below), and enter mode II, III, or IV, depending on the outcome of this assessment.

Narrowing down the set of potentially faulty nodes: Each node forms a *diagnosis graph* after the broadcast phase in *extended round 3*, as elaborated next. The diagnosis graph contains four vertices

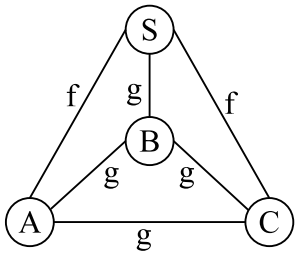


Figure 3: Diagnosis Graph

S, A, B and C, corresponding to the four nodes in our network; there is an undirected edge between each pair of vertices, with each edge being labeled as g at time 0 (with g denoting "good"). The labels may change to f during extended round 3. Once a label changes to f (denoting "faulty"), it is never changed back to g. Without loss of generality, consider the edge between vertices X and Y in the diagnosis graph. The label for this edge may be set to f in two ways:

- After the broadcast phase, all nodes obtain identical information about what every node "claims" to have sent and received during rounds 1 and 2. Then, for each packet in rounds 1 and 2 (sent by any node), each fault-free peer will compare the claims by nodes X and Y about packets sent and received on links XY and YX. If the two claims mismatch, then the label for edge XY in the diagnosis graph is set to f.

- If node X is a peer and claims to have detected a misbehavior in round 3, but the packets it claims to have received in rounds 1 and 2 are inconsistent with this claim, then edge XY in diagnosis graph is set to f. In this case, all edges associated with X are set to f.

Similar actions are taken for each of the 6 undirected edges in the diagnosis graph. An example diagnosis graph thus obtained is illustrated in Figure 3. Due to the use of Byzantine agreement for the broadcast phase, all nodes will form identical diagnosis graphs. The notion of diagnosis here is borrowed from past literature on *system-level diagnosis* [23], and the notion on conflict graphs in the work on continuous consensus [20].

It should not be difficult to see that the edge between vertices corresponding to fault-free nodes will remain g. For instance, if nodes A and B are fault-free, then edge AB in the diagnosis graph will remain g. Now we argue that at least one link between the faulty node and a fault-free node will be f after a failure detection. We will consider the case when S is faulty and a peer is faulty separately.

A peer is faulty: Without loss of generality, suppose that node A is faulty. It misbehaves either by (i) raising a false alarm (implicitly accusing another node of misbehavior) or (ii) by sending some tampered packets to nodes B or C. Let us consider each case. (i) If node A broadcasts only correct packets in the broadcast phase, that will contradict with the false alarm, and edges BA, CA and SA in the diagnosis graph will all be labeled f. On the other hand, if node A broadcasts incorrect packets, inconsistent with packets actually received from node S, then edge SA will be labeled f (note that when A is faulty, S is fault-free). (ii) Now suppose that node A has misbehaved by sending incorrect packets to another peer: In this case, if A broadcasts correct packets in the broadcast phase, the edge between the recipient of the tampered packets from A (in round 2) and node A will be marked f. Otherwise, edge SA will be marked f similar to case (i).

Node S is faulty: Node S can misbehave only by sending $k + l + m$ packets in round 1 such that all subsets of R packets do not have a unique solution. During the broadcast phase, if S broadcasts packets such that subsets of R packets do not have a unique solution, then it is clear that node S is misbehaving, and edges SA, SB and SC in the diagnosis graph are all marked f. On the other hand, if

the packets broadcast by S in the broadcast phase all have a unique solution, then the packets received by at least one peer in round 1 will differ from packets sent by S in the broadcast phase. Thus, the edge between that peer and S in the diagnosis graph will be marked f.

Now it should be not hard to see that the faulty node must be one of the two nodes associated with an f edge in the diagnosis graph. So, if there is only one f edge, we can narrow down the faulty node to be **one** of the two nodes corresponding to the two endpoints of that edge – the set of these two nodes is called the "fault set". If there is more than one f edge, then they must share a vertex in common, and the node corresponding to the common vertex must be the faulty node (recall that edge between vertices for two fault-free nodes can never become f). For instance, the diagnosis graph in Figure 3 implies that node S must be faulty. Depending on the outcome of this "diagnosis" using the diagnosis graph, the system enters mode II, III, or IV. In particular, when the "fault set" is narrowed down to two peers, the system enters mode II; if the fault set contains a peer and node S, the system enters mode III; and if the fault set contains only one node (then the faulty node is known exactly to all other nodes), the system enters mode IV.

How much additional time is required for the *extended round 3*? Since we assume that the capacity of all links is > 0, it follows that the broadcast phase in extended round 3 will require a finite amount of time, although the duration would be large when link capacities are small. In particular, the number of bits that needs to be transmitted on each link during extended round 3 is $O(Rc)$, and with non-zero capacity on each link, the time required would be $O(Rc)$ as well. As we will see later, the broadcast phase occurs only once for each mode change, and mode change occurs at most twice. Thus, as we will also see later, in a pipelined execution, the negative impact (on the throughput) of extended round 3 becomes negligible as time increases.

7.2 Operation in Mode II

Mode II is entered when the fault set is narrowed down to two peers. Without loss of generality, assume that the location of the faulty node is narrowed down to the set {A,B}, and node A is actually the faulty node. Recall that the fault set is {A,B} when only edge AB in the diagnosis graph is f. This also implies that node C does not know the exact identity of the faulty node, but knows that node S is definitely fault-free. Fault-free node B knows that nodes C and S are both fault-free. Round 1 in mode II is same as that in mode I. The schedule in round 2 in mode II is similar to round 2 in mode I, with the only change being that no packets are scheduled on links AB and BA (any packets received on these links are ignored).

Since node B knows that nodes S and C are fault-free, and since it received at least R packets from S and C together (this follows from condition SC1), node B can use the packets received from S and C to recover the correct data sent by node S in round 3.

In round 3, node C attempts to find an unique solution that is consistent will all the packets it receives from nodes S, A and B. (Recall that each packet is supposed to be a linear combination of the data that the sender wants to send, and any set of R of these linear combinations should be independent provided that no node has misbehaved.) If node C finds a unique solution, it must be identical to $\tilde{x}$ sent by node S (since node C receives at least R correct packets from S and the fault-free peer) – in this case, node C agrees on the unique solution. If there is no such unique solution, node C has detected tampering of packets, although it does not yet know that node A is the faulty node. Similar to mode I, the failure detection is disseminated to other nodes using the traditional Byzantine agreement algorithm. If and only if node C has detected a failure, *extended round 3* is performed similar to extended round 3 in mode I (that is, *broadcast phase* is performed to reliably broadcast all packets sent/received in rounds 1 and 2).

Narrowing down the set of faulty nodes: During the broadcast phase, node A has two choices: (i) Corresponding to the tampered packets it sent to node C in round 2, node A may broadcast identical

(tampered) packets in the extended phase 3. In this case, these packets will mismatch with packets S claims to have sent to A in round 1, resulting in edge SA in the diagnosis graph being set to f. (ii) Alternatively, node A can send correct packets during the broadcast phase. In this case, there will be a mismatch between packets that A claims to have sent in round 2, and C claims to have received in round 2, resulting in edge AC in the diagnosis graph being set to f. In cases (i) and (ii) both, since edge AB in the diagnosis graph was already f, we will now have two edges f, both of which share vertex A. This implies that node A must be faulty, and all fault-free node learn the identity of the faulty node. At this point, the system has transitioned to mode IV from mode II.

7.3 Operation in Mode III

Mode III is entered when the fault set is narrowed down to a set containing node S and one peer. Without loss of generality, assume that the location of the faulty node is narrowed down to the set {S,B}. Recall that the fault set is {S,B} when only edge SB in the diagnosis graph is f. In this case, nodes A and C know that they are both fault-free.

In mode III, the schedule in round 1 is the similar to that in mode I, with the only difference being that no transmissions are scheduled on link SB (any packets on link SB are ignored). The schedule in rounds 2 and 3 needs to modified slightly compared to mode I. We discuss rounds 2 and 3 for the following two cases separately: (a) $AC + CA \geq R$, and (b) $AC + CA < R$.

(a) $AC + CA \geq R$**:** In round 2, schedule as many distinct packets received from S in round 1 as possible on links AC and CA (node B is not to transmit in round 2, and packets received from B may be ignored). Similar to the case of $AC + CA > R$ in the proof of Theorem 1, in this case as well, nodes A and C exchange at least R coded packets on links AC and CA in round 2. In round 3, nodes A and C try to find the unique solution for the packets they have received from each other and from node S. If A or C cannot find a unique solution, then it follows that node S must be faulty, appropriate 1-bit notifications are reliably broadcast to the other nodes at the end of round 3, and the system then transitions to mode IV. If nodes A and C are both able to find a unique solution in round 3, they agree on that unique solution consisting of R data packets $\tilde{x}$. Also in round 3, nodes A and C together send these R data packets to node B on links AB and CB. According to SC2, this is always possible. If B happens to be fault-free, it agrees on the data packets received from nodes A and C. Note that in this case, extended round 3 is not necessary.

(b) $AC + CA = R - \delta < R$**:** In round 2, nodes A and C exchange $R - \delta$ packets on links AC and CA (this also follows from the arguments used in the proof of Theorem 1). Note that A receives at least R packets from S and C, but exchanges only $R - \delta$ with C in round 2, which implies that A has at least δ packets not yet shared with C. Similarly, C has at least δ packets not yet shared with A. Since $AB + CB \geq R \geq \delta$, in round 2, it is possible to schedule on links AB and CB **together** δ packets, which are distinct from the $R - \delta$ packets exchanged on AC and CA in round 2. In round 3, A and C together send to B **all** the $R - \delta$ packets they have previously (in round 2) exchanged with each other. Observe that the total number of packets sent on links AB and CB in round 2 and 3 together is R, and $R < AB + CB$ (by condition SC2).

Now, since $AC + CA = R - \delta$, and $BA + CA > R$ (by sufficient condition SC2), it follows that $BA > \delta$. By a similar argument, $BC > \delta$. In round 3, node B schedules on each of links BA and BC the δ packets it receives in round 2 from A and C. Note that if node B does not misbehave then A and C are now guaranteed to have R identical packets among the packets they have received in rounds 1 through 3. Similarly, if S does not misbehave, A and C will have R correct packets (by condition SC1). If node A or C cannot find a unique solution that is consistent with all the received data packets, then

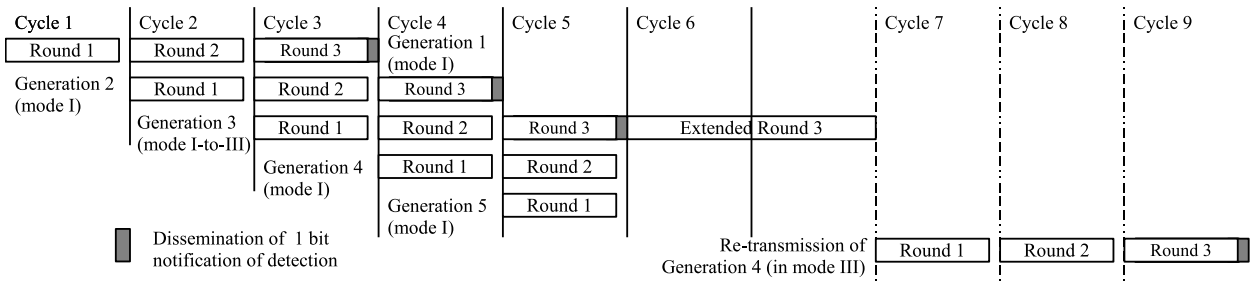


Figure 4: Example of pipelining: In generation 3, node A detects a mismatch and the system enters mode III. Generations 4 and 5 are then dropped and retransmitted after entering mode III.

it has detected packet tampering. In this case, similar to modes I and II, after distributing the 1-bit notifications, the system will enter extended round 3, and the nodes perform the broadcast phase.

Since the fault set is {S,B}, either node S or node B must have tampered some packets. Similar to mode II, in this case as well, after the broadcast phase, one more edge in the diagnosis graph will become f. The common vertex to the two f edges in the diagnosis graph will indicate the faulty node (also similar to mode II), and the system will transition to mode IV.

7.4 Operation in Mode IV

When the network is in mode IV, the identity of the faulty node is correctly known to all other nodes. In the event that node S is known to be faulty, the fault-free peers can decide to agree on some default value, and terminate the algorithm. In the event that a peer is known to be faulty, the fault-free nodes ignore all packets received from the faulty peer. In this case, in rounds 1 and 2, the schedule remains the same as in rounds 1 and 2of mode I. Each fault-free peer can recover $\tilde{x}$ using the packets receives from the fault-free source and the other fault-free peer (by condition SC1).

7.5 Throughput Analysis

With the exception of the extended round 3, each round in our algorithm uses identical time, which is slightly longer than c time units (why the round length is slightly longer than c will be clear soon). Observe that with the exception of the dissemination of 1-bit notification and the broadcast phase (in extended round 3), the usage of each link is within the link capacity in each mode. In particular, a link with capacity z carries at most z **data** packets combined over all rounds in each mode (ignoring the extended rounds). In achieving rate R, it will be necessary to have multiple "generations" of packets in the network, with the algorithm operating in a pipelined manner (one round per pipeline stage). Agreement algorithm for one new generation of data of size Rc bits (or R symbols from GF(2^c)) starts per "clock cycle", as shown in Figure 4. Each generation consists of three rounds and the packets are scheduled according to the schedules we discussed above. By the end of the round 3 of every generation, the peers exchange 1-bit notifications indicating whether they detected an attack. Subsequently, if necessary, an extended round 3 is performed. By the end of the extended round 3, if any, all nodes decide on the same new mode.

Figure 5 shows an example execution of the pipelining. The system starts in mode I. Nominally, 1 clock cycle is assigned to each round, including round 3 (when it is not extended). When round 3 is not extended, much of the time in round 3 for modes I, II and IV would be idle time. In the example execution, the system is in mode I for generations 1 and 2. During round 3 of generation 3, node A detects packet tampering, and round 3 is extended as shown – in this case, the extended round 3 happens to require 2 additional cycles (but this could be larger in general). By the end of this extended

round 3 (cycle 6), the identity of the faulty node is confined to {A,S} and the system enters mode III. Note that, at the time the system enters mode III, two generations of packets are in the system using the old schedule for mode I (in this example: generations 4 and 5). To allow a transition to the new schedule in mode III, the packets belonging to generations 4 and 5 are dropped, and retransmitted using the algorithm/schedule for mode III. Thus, agreement for the dropped generations is re-initiated in subsequent clock cycles.

Recall that there are at most two mode transitions throughout the execution of the algorithm (mode I to II to IV, or mode I to III to IV), thus requiring at most two extended rounds. The time overhead of an extended round 3 is $O(Rc)$ – during the extended round, no new generations make progress. Since the extended round occurs at most twice, the average overhead *per cycle* of extended round 3 decreases to zero as time increases. In particular, the overhead per cycle becomes $O(\frac{1}{Rc})$ after R^2c^2 cycles. The dissemination of 1-bit notifications (in round 3) requires a fixed number of bits per link, independent of R and c. Thus, the total time overhead for this operation is $O(1)$. Thus, we can make the duration of each round to be equal to $c + O(1)$. Since a new generation of Rc bits worth of information is initiated in each round, it follows that the agreement throughput is $R - O(1/c)$. Thus, by using a suitably large c, the throughput can be made arbitrarily close to R.

Thus, we have shown that conditions SC1 and SC2 are sufficient for achieving agreement throughput approaching R bits/unit time in a complete four-node network.

8 Byzantine Agreement with One Uplink

It turns out that SC1 and SC2 together are also sufficient to achieve throughput arbitrarily close to R in four-node networks with two or one uplink, in presence of a single Byzantine failure.

In this section, we present a Byzantine agreement algorithm for four-node networks with One uplink (BAO) that achieves throughput arbitrarily close to R bits/unit time, given conditions SC1 and SC2 are satisfied. The achievability for networks with two uplinks naturally follows by using only one uplink. Also notice that BAO also achieves capacity in complete four-node networks as BAC. Similar to BAC, BAO operates in rounds and has the same four modes of operation.

8.1 Operation in Mode I

Rounds 1 and 2: Use the same coding strategy and schedule as rounds 1 and 2 in BAC.

Round 3: The operation is almost the same as round 3 in BAC. Each fault-free peer finds the solution for **each** subset of R packets from among the packets received from the other three nodes in rounds 1 and 2. Let X be the peer with the uplink and Y, Z be the other two peers with no uplink. Y (Z) first sends a 1-bit notification indicating whether it has detected a failure (1) or not (0) to X and Z (Y). Then Y (Z) forwards the notification it just received from Z (Y) to X.

After this, X has received 4 bits from Y and Z: 1 bit from link YX directly, denoted as N_{YX}; 1 bit from Y via Z, denoted as N_{YZX}; 1 bit from link ZX directly (N_{ZX}); and 1 bit from Z via Y (N_{ZYX}). Then X broadcast the four notifications from Y and Z and its own notification N_X (5 bits in total) using a scheme similar to the traditional Byzantine agreement algorithm. Since there are three node-disjoint paths from X to each of Y and Z and only one node may misbehave, it is guaranteed that Y and Z will reach an agreement on the notification bits using majority vote.

At the end, node S broadcasts a 1-bit confirmation as the OR of the 5-bit notifications it received from X, using the traditional Byzantine agreement algorithm. This broadcast from S is reliable since there are three node-disjoint paths from S to every peer.

If a fault-free peer Y detects a failure, it sends $N_{YX} = N_{YZ} = 1$. In the case the 5-bit notifications broadcast by X are all 0's, it is clear to both Y and Z that X is faulty, and they enter mode IV together with the fault set {X}. (The procedure for notification ensures that if any fault-free node sends/receives 1, every peer gets 1.)

If any one of the 5-bit notifications broadcast by X is 1, but S replies with a 0 by the end of round 3, it is clear to the fault-free peers that the faulty node must be either S or X. Then the fault-free peers Y and Z both enter mode III with the fault set {S,X}. In the case X is actually fault-free, the 5-bit notifications it sends to S are the same as the ones Y and Z receives. So at least one of the notifications S receives from X must be 1, but S does not confirm that. So X can be sure that S is faulty, and X would know that other fault-free peers (Y and Z) are switching to mode III. Then X will enter mode III too, to help Y and Z with their communication, as elaborated later.

In the case that S replies with a 1, round 3 is extended.

Extended Round 3: In extended round 3, the nodes exchange packets that have been sent and received during rounds 1 and 2, and decide on the new mode, following the schedule below:

1. S broadcasts all the packets it has sent in round 1, using the traditional Byzantine agreement algorithm. This broadcast is reliable since there are three node-disjoint paths from S to each peer. The fault-free peers will agree on identical packets from S.

2. Each peer sends the packets it has received and sent in rounds 1 and 2 to the other two peers (only on the direct links). So after this, each peer X receives two copies of packets that have been transmitted on link YZ in round 2: one from node Y about what it has sent to Z, and the other one from node Z about what it has received from Y. In the rest of this section, we will refer to steps 1 and 2 as the broadcast phase in BAO. Based on the notifications received in round 3 and the data packets received in the broadcast phase, each peer identifies the *trusted set* of fault-free nodes (as elaborated later).

3. Each peer sends the trusted set to the other peers, on the direct links.

4. Based on the trusted set received from the other two peers, each peer updates its own trusted set (as elaborated later). Then the peer decides on the new mode based on its own trusted set and sends the mode index together with its trusted set to the other two peers.

5. If a peer decides on mode IV, but the mode index it receives from a trusted peer is not IV, it changes its own mode to follow the trusted peer's mode.

We will say that a peer X *trusts* a node Y, denoted as X→Y, if X is sure that Y is fault-free. Now we present an algorithm for each node to decide on the trusted set individually, such that after step 2 of extended round 3, a fault-free peer will not trust a faulty node, and will trust at least one of the other fault-free nodes.

Algorithm to form the trusted set: We consider the cases of faulty source and faulty peer separately.

Node S is faulty: Node S can misbehave either (i) by starting a broadcast even if no failure is detected by the peers, or (ii) by sending $k+l+m$ packets in round 1 such that all subsets of R packets do not have a unique solution. Without loss of generality, assume that peer X is the one with the uplink.

Using the same notation X, Y and Z as before, in (i), the 5-bit notifications broadcast by X are all 0's, but S replies with a 1 and enters extended round 3. It is clear to X that S is faulty, and it

is clear to Y and Z that either S or X is faulty. Then X→{Y,Z}, Y→Z and Z→Y, without using the information exchanged during the broadcast phase.

In (ii), at least one of the peer detects the failure. Then at least one of the 5-bit notifications broadcast by X is 1. The case when S replies with a 0 has been discussed at the second last paragraph of round 3. Now we consider the case when S replies with a 1 and starts the broadcast phase. If S broadcasts packets such that subsets of R packets do not have a unique solution, then it is clear to all peers that node S is misbehaving, so every peer trusts the other two peers. On the other hand, if the packets broadcast by S in the broadcast phase all have a unique solution, then the packets received by at least one peer, say A, in round 1 will differ from packets sent by S in the broadcast phase. Then A identifies the faulty node as S, hence A→B and A→C. Meanwhile, the packets S broadcasts also mismatch with the packet A claims to B and C that it receives in round 1. Then B and C can both form their local fault sets as {A,S}. Thus B→C and C→B.

In both cases, S is not trusted by any fault-free peer, and every peer trusts at least one other peer, which is fault-free. Also notice that at least one peer identifies the faulty node as S in all cases. If exactly one peer identifies S as faulty, the other two peers trust each other. This fact will be used later.

A peer is faulty: Without loss of generality, suppose that node A is faulty. It misbehaves either (i) by tampering the notifications; (ii) by sending some tampered data packets to nodes B or C. We will consider these two scenarios separately.

(i) Node A tampers the notifications:

- If A is the node with the uplink, then A broadcast the 5-bit notifications to B and C. If the notifications broadcast by A are all 0's, but S replies with a 1, it is clear to B and C that the faulty node must be either S or A. Then the fault-free peers B and C will trust each other, without using the information exchanged in the extended round 3. The case when one of the notifications A broadcasts is 1, but S replies with a 0 has been discussed in the second last paragraph of round 3.

  Now we consider the cases when the reply from S is consistent with the notifications X broadcasts:

  - Node A tampers broadcast of N_{BA}: it is clear to B that A is faulty, hence B→{S,C}. And since the N_{BA} broadcast by A mismatches N_{BC} – the notification C receives from B directly in round 3, C can be sure that either A or B is faulty. Then C→S.
  - Node A tampers broadcast of N_{BCA}: it is clear to C that A is faulty, hence C→{S,B}. And since the N_{BCA} broadcast by A mismatches the N_{BC} that B sent to C in round 3, B can be sure that either A or C is faulty. Then B→S.
  - Node A tampers broadcast of N_{CA} or N_{CBA}: similar to the first two cases.
  - Node A broadcasts $N_A = 1$: if S replies with a confirmation, extended round 3 is entered. During the broadcast phase, if A sends to B data packets that are different from the ones A receives from S in round 1, B sees a mismatch between A and S, and B→C. If A sends to B data packets that are different from the ones A receives from C in round 2, B sees a mismatch between A and C, then B→S. If A sends to B data packets as it receives in rounds 1 and 2, these packets should be correct and consistent (since S, B and C are fault-free). So B is sure that A has incorrectly sent $N_A = 1$. Then B→{S,C}. Similar for node C.

- If A is a node without the uplink, suppose that B is the peer with the uplink. A can raise a false alarm by sending $N_{AB} = 1$ or $N_{AC} = 1$. In this case, C forwards $N_{ACB} = N_{AC}$ correctly and B broadcasts them correctly. And then S will start the broadcast. Two cases are possible:

- $N_{AB} \neq N_{AC} = N_{ACB}$: it is clear to B that either A or C is faulty, hence B→S. Similarly, it is clear to C that either A or B is faulty, hence C→S.
- $N_{AB} = N_{AC} = N_{ACB} = 1$: Similar to the case when A has the uplink and broadcasts $N_A = 1$.

When A is a node without the uplink, it can also tamper the notification from C by sending $N_{CAB} \neq N_{CA}$ to B. In this case, since $N_{CB} = N_{CA} \neq N_{CAB}$, it is clear to B that either A or C is faulty, hence B→S. It is also clear to C that either A or B is faulty, thus C→S.

(ii) Node A sends tampered data packets to the peers: Without loss of generality, suppose that A sends tampered data packets to B in round 2. In this case, B must see a mismatch in the packets from A and the packets that S broadcasts in extended round 3; thus B suspects {S,A}, and B→C. Remember that we are looking at the case when round 3 is extended. The case when round 3 is not extended has been discussed previously in round 3. During the broadcast phase, if A sends to C correct packets that it should have sent to B in round 2, C sees a mismatch in these packets and the packets it receives from B in step 2 of the broadcast phase, then C→S. If A sends to C incorrect packets different from the ones it should have sent to B, C sees a mismatch in these packets and the packets S broadcasts in step 1 of the broadcast phase, then suspects {S,A}, and C→B. It is similar if A sends tampered data packets to C.

From the above discussion on the algorithm, we conclude the following theorem

Theorem 2 *After step 2 of extended round 3, a fault-free peer will not trust a faulty node, and will trust at least one of the other fault-free nodes.*

Now we have proved in the extended round that a fault-free peer will never trust a faulty node after step 2. Which implies the following corollary (under single failure assumption):

Corollary 1 *If X claims to Y in step 3 that X→Z (X≠Z), Z must be fault-free.*

After exchanging the trusted sets in step 3, each peer can *expand* its own trusted sets by trusting the nodes trusted by the other two peers.

According to the discussion on algorithm for forming a trusted set, it should be easy to see that if the source is faulty, at least one peer will identify that S is faulty after step 2. If only one peer, say A, identifies S as faulty after step 2, then B and C will trust each other as per discussion above in the case of faulty source. In this case, the trusted set of each peer remains unchanged after step 4, and A is in mode IV and the other two peers in mode III with fault set {S,A}. In step 5, A copies B and C, then the nodes enter mode III after step 5, with the fault set {S,A}. If two peers identify S as faulty after step 2, without loss of generality, assume that A→{B,C} and B→{A,C}. Then after in step 4, C will trust B since A→B. C will also trust A since B→A. So C can conclude that S is actually faulty. And all peers are in mode IV, with the fault set {S}.

Now let us consider the case when a peer is faulty. Without loss of generality, assume that node A is faulty. According to Theorem 2, B will trust S or C, and C will trust S or B in step 3. And according to Corollary 1, their trusted sets may be expanded in step 4. Table 1 enumerates all possible combinations of initial trusted sets of B and C, and the corresponding expanded trusted sets if A's trusted sets are ignored. Notice that no matter what A claims about its trusted sets to B and C, the expanded trusted sets of B and C either are the same as listed in Table 1, or one fault-free node is added into the expanded set.

So in cases 1 of Table 1, the combination of the expanded trusted sets of B and C falls into one of cases 1, 3, 4 and 5 in Table 1, depending on what A claims to B and C.

	Initial (step 3)		Expanded (step 4)		
Case	Node B	Node C	Node B	Node C	Modes of B and C
1	S	S	S	S	II
2	C	B	C	B	III
3	S	B	S	S, B	II-IV
	S	S, B	S	S, B	II-IV
4	C	S	S, C	S	IV-II
	S, C	S	S, C	S	IV-II
5	C	S, B	S, C	S, B	IV
	S, C	B	S, C	S, B	IV
	S, C	S, B	S, C	S, B	IV

Table 1: Initial trusted set at nodes B and C when A is faulty. And the corresponding expanded trusted sets if A's trusted sets are ignored.

In case 2, B may trust S if A claims to B that A→S. Then the B enters mode IV. Similar if A claims to C that A→S. And the modes of B and C may end up in IV-III, III-IV, or IV.

In cases 3 and 4, if the trusted sets of B and C are further expanded by A's trusted set, they all enters mode IV.

In the case of mode II-IV, node C follows B and change its mode to II with fault set {A,B}. Similar for the case of modes IV-II, III-IV, and IV-III.

In summary, in mode I, if any one node misbehaves, it will be detected, and the fault-free peer will enter mode II, III, or IV together.

8.2 Operation in Mode II

Mode II is entered only when a peer is faulty, and both fault-free peers trust only S after the extended round 3. Without loss of generality, assume that A is faulty. The operation in mode II is the same as mode I, with following two new rules on how to handle the notifications of detection:

1. A peer will not raise an alarm once it identifies the faulty node (fault set contains only one node). In other words, once a peer identifies the faulty node during a round 3, the notifications it sends in future round 3's are always 0's.

2. Let X be the peer with the uplink and Y, Z be the other two peers. X always broadcasts $N_{YZX} = N_{ZYX} = 0$. If the broadcast phase in a round 3 is triggered after X broadcasts $N_{YX} = 1$, X will ignore the incoming N_{YX} and broadcast $N_{YX} = 0$ in future round 3's. Same rule applies to N_{ZX}.

The reason for the rule 1 is that once a fault-free peer has identified the faulty node, it can always recover the data correctly from the packets received from the other two fault-free nodes and agree with them. And the reason for rule two is that in mode II, all fault-free peers trust S. If a fault-free peer detects a failure and the broadcast phase is entered, it knows which peer is faulty right away by comparing the packets S broadcasts and the ones it receives in round 2. Rule 2 in fact limits the number of broadcast phases the faulty peer can trigger to be at most three in mode II.

We consider the cases when node A tampers data packets and notifications separately.

Node A sends tampered data packets: If A sends tampered data packets to B, B will detect an attack. If A is the node with the uplink, and S replies with a 0, B identifies A as faulty, since B is

sure that S is fault-free. If S confirms with an 1 and starts the extended round 3, B sees a mismatch between packets received from A and the packets S broadcasts. Then B identifies A as the faulty node. Similarly, if A sends tampered packets to C, C will identify A as the faulty node.

Node A tampers the notifications:

- If A is the node with the uplink, then A broadcasts the notifications from B and C, and its own 1-bit notification. If the reply from S is inconsistent with the notifications A broadcasts, it is clear that either A or S is faulty. Remember that both B and C are sure that S is fault-free. Then B and C both identify A as faulty and enters mode IV. In the case that the reply from S is consistent with the notifications A broadcasts:
  - Node A tampers broadcast of N_{BA}: it is clear to B that A is faulty, hence B→{S,C}, and B enters mode IV.
  - Node A tampers broadcast of N_{CA}: it is clear to C that A is faulty, hence C→{S,B}, and C enters mode IV.
  - Node A broadcasts $N_{BCA} = 1$ or $N_{CBA} = 1$: it is clear to B and C that A is faulty since a fault-free A should always broadcast $N_{BCA} = N_{CBA} = 0$. Hence B→{S,C} and C→{S,B}. Then both B and C enters mode IV.
  - Node A broadcasts $N_A = 1$: It is possible that B and C will remain only trusting S after the extended round 3. So they stay in mode II. However, notice that in mode II, if a fault-free node detects a failure, it always identifies the faulty node, and will not raise an alarm any more. So in both B's and C's point of view, if A is fault-free, A must have identified the faulty node after the broadcast phase triggered by $N_A = 1$. Then if A broadcasts $N_A = 1$ again, B and C both identify A to be the faulty node, and both enter mode IV.
- If A is a node without the uplink, without loss of generality, assume that B has the uplink. Node A can raise a false alarm by sending $N_{AB} = 1$. In this case, B will broadcast $N_{AB} = 1$ correctly and S will confirm. It is possible that nodes B and C still cannot identify the faulty node after the broadcast phase. However, similar to the case when A has the uplink, if A sends $N_{AB} = 1$ or $N_{AC} = 1$ again, B or C will be able to identify A as faulty. Notice that if A tampers N_{CAB} or just sends $N_{AC} = 1$, no broadcast will be triggered, according to rule 2.

In any case, in mode II, either the two fault-free peers can agree at rate R, or the faulty peer can trigger the broadcast phase for at most three times before it is identified by both of the two fault-free peers and the system enters mode IV.

8.3 Operation in Mode III

Mode III is entered when two fault-free peers trust each other, and at least one of them does not trust the third peer nor the source. Without loss of generality, assume that A→C and C→A. If B is in fact fault-free, it is aware that S is faulty and should follow nodes A and C into mode III.

The operation of mode III is the same as in BAC, except for the handling of notifications of detection in round 3 and the broadcast phase in extended round 3.

In round 3, the dissemination of the notifications are the same as in mode I, except that B should always claim it detects no attack, and the peer with the uplink should always broadcasts all 0's for the 5-bit notifications. In the case A or C detects an attack, they send $N_{AC} = N_{AB} = 1$ or $N_{CA} = N_{CB} = 1$. The extended round 3 is entered by the peers only when $N_{AC} = N_{AB} = 1$ or $N_{CA} = N_{CB} = 1$.

If any one of the notifications B sends out is 1, A and C both identify B as faulty and enter mode IV (may be after extended round 3).

In the case $N_{AC} = N_{AB} = 1$ or $N_{CA} = N_{CB} = 1$, the extended round 3 is entered by the peers. Nodes A and C exchange on links AC and CA directly all the data packets received in rounds 1, 2 and 3. Similar to the discussion in BAC, both A and C will be able to identify the faulty node after this, and they will transition to mode IV. The pipelining will need to be modified, since S will not stop sending new generations of data packets when the peers enter the extended round 3: nodes A and C buffer the new generations received from S during the broadcast phase, and node B will "wait" for them until the extended round 3 finishes. Then the peers perform a "delayed" version the agreement algorithm on the buffered generations.

8.4 Operation in Mode IV

When the network is in mode IV, the identity of the faulty node is correctly known to all other nodes. The operation is the same as in BAC.

8.5 Throughput Analysis

As we have seen in the previous subsections, at most four broadcast phases can be triggered throughout the execution of the algorithm. In the case that S is fault-free and receives notifications as 1's in round 3 for more than four times, it is clear to S that the peer with the uplink is faulty and will ignore further notifications indicating failure detected. On the other hand, if node S is faulty, and start the broadcast phase for more than four times, all the peers will know that S is faulty and enter mode IV.

To achieve throughput arbitrarily close to R, BAO is executed in a pipelined fashion similar to BAC. From the above discussion, we can see that the broadcast phase in extended round 3 will be performed for at most four times. So according to the same argument for BAC, the overhead for the broadcasting in BAO also diminishes to zero, which implies the achievability of throughput R.

9 Byzantine Agreement with No Uplink

In the previous sections, we have shown the sufficiency of SC1 and SC2 in four-node network with at least one uplink. Since SC1 and SC2 can be arbitrarily close to NC1 and NC2, it follows that the capacity of a four-node network with uplink(s) is the supremum over all throughputs R that satisfy NC1 and NC2. However, it turns out that the agreement capacity is changed dramatically when there is no uplink. Since in this case, NC4 must also be satisfied.

- **Sufficient condition SC4:** The capacity of any downlink from S must is $> R$, when there is no uplink.

It turns out that SC4 is sufficient together with SC1, SC2 and NC3. The achievability is proven by BAO with this slight modification: In round 1, instead of generating $k + l + m$ packets, node S just sends the same R data packets to each of the peers. According to SC4, there is enough capacity on links SA, SB and SC. In round 3, each peer's notifications are relayed through the two cycles: A-B-C-A and A-C-B-A. By doing so, it is guaranteed if a peer claims an attack detected, all peers will be notified. Since A, B and C have already received all the R data packets from S in round 1, they just send these data packets to each other and take a majority vote, and a reliable broadcast from S is achieved. And the rest of the algorithm remains the same as BAO.

The pipelining will need to be modified too, since S will not stop sending new generations of data packets when the peers enter the broadcast phase: the peers buffer the new generations received from S during the broadcast phase, and perform a "delayed" version the agreement algorithm on the buffered generations. This may introduce extra decoding delay. However, the throughput is not affected.

The agreement capacity of a four-node network with no uplink is the supremum over all throughputs R that satisfy necessary conditions NC1, NC2, NC3 and NC4.

10 Delay in Achieving Agreement: Impact of Link Usage

As we have seen, to achieve capacity, multiple instances of an agreement algorithm may need to overlap in time. Let us consider agreement algorithms wherein the data packets forwarded in different instances of the algorithm do not depend with each other. Thus, each instance allows the fault-free nodes to agree on different "generations" of data. For such algorithms, we prove the following lower bounds on the number of rounds of messages required to reach agreement on each generation of data.

Theorem 3 *To achieve agreement on σ bits, at least 3 rounds of message exchange are necessary, if less than σ bits are sent on any incoming link at any peer.*

As elaborated in Appendix A.1, this implies the necessity of at least 3 rounds of message exchange per generation to achieve agreement throughput R approaching capacity, if an incoming link at a peer is less than R. We also prove the theorem below for the case when the links between a pair of peers have limited capacity. Our capacity achieving algorithm with a slight optimization can be made optimal in terms of average number of rounds of message exchange. The proof of the two theorems and details of optimizing our algorithm can be found in Appendix A.1, A.2, and A.3.

Theorem 4 *To achieve agreement on σ bits, at least 4 rounds of message exchange are necessary, if less than σ bits are communicated through links between any one pair of peers.*

11 An Alternative Algorithm for Complete Four-Node Networks with Lower Computational Complexity

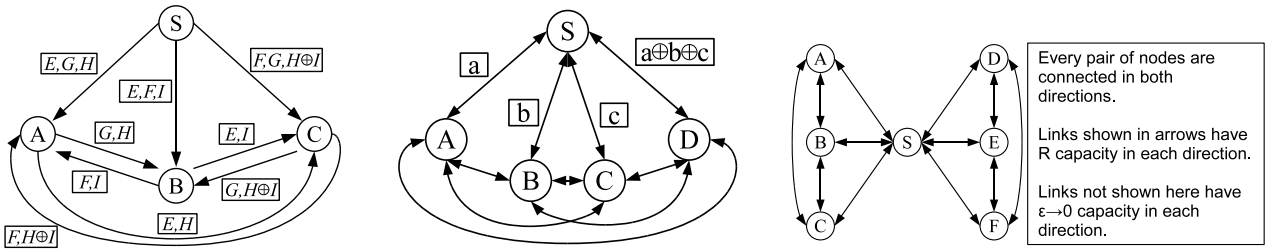
The Byzantine agreement algorithms in previous sections requires every peer to find solutions for many sets of R linear equations, which can incur heavy computational overhead. In this section, we present an capacity achieving algorithm for the complete four-node networks in which the peers only need to perform simple bit-wise operations.

It is easy to see that, if the necessary conditions NC1 and NC2 hold for a certain value of R, then they also hold if we replace each link XY with another link of capacity $minimum(R, XY)$, where XY denotes the original capacity of link XY. Therefore, to simplify the discussion below, we assume that each link in our network has capacity at most R.

Assume that the length of each round is (slightly more than) 1 time unit. Similar to the BAC, there are four modes of operation. For brevity, we present the schedule only for mode I of this algorithm. Assume that sufficient conditions SC1 and SC2 are true.

Round 1: Sender S transmits R bits of data to nodes A, B and C as follows. The R bits of data is divided into 6 parts or "packets", named D through I, with the following sizes (refer to Figure 1(b) for definition of rates k, l, m): packet D of size $d = \max\{k+m+l-2R, 0\}$, packet E of size $e = k+l-R-d$, packet F of size $f = l+m-R-d$, packet G of size $g = k+m-R-d$, and packets H and I both of size $h = d-k-m-l+2R$. Packet $H \oplus I$ is computer as a bit-wise exclusive OR of packets H and I. Note that the sizes of packets D, E, F, G, H, I add up to R. In round 1, node S transmits packets as follows: D, E, G, H to node A, D, E, F, I to node B, and $D, F, G, H \oplus I$ to node C.

Observe that when $k+m+l \leq 2R$, $d = 0$. In this case, packet D is non-existent. On the other hand, when $k+m+l > 2R$, packet D has a non-zero size, but $h = 0$. For brevity, we present here only the schedule when $k+m+l \leq 2R$. For the details of the case when $k+m+l > 2R$, the reader



(a) $k + l + m \leq 2R$

(b) 5-node complete network. Each directed link has capacity 1

(c) Conditions SC1 and SC2 are not sufficient for the seven node network

Figure 5: Alternative algorithm for the complete four node network and examples of general networks

is referred to our technical report [17]. The packets transmitted by S when $k + m + l \leq 2R$ are shown in Figure 5(a).

Round 2: Recall that we assume $k + m + l \leq 2R$. Let us consider peer A. Peer A transmits to peer B packets that A has received from S, but B has not. Specifically, node A forwards packets G, H to node B. Other peers similarly forward packets to the other two peers. The resulting schedule of transmission on the links between the peers is also shown in in Figure 5(a).

Round 3: In round 3, the peers use the packets they have received during round 1 and 2 to identify any inconsistencies (using a few bit-wise XOR operations). The rest of the features of the algorithm, including 1-bit notifications, extended round 3, and narrowing of the fault location to a subset of nodes, are similar to the previously discussed algorithm. Please refer to [17] for details.

12 Oblivious Algorithms

In the previous algorithms , the schedule of packets changes over time. In this section, we briefly sketch an oblivious algorithm for complete four-node networks when the following two conditions are satisfied in addition to SC1 and SC2.

- **Condition SC3:** The sum capacity of the links between every pair of nodes is $> R$.
- **Condition SC4:** The min-cut from S to every peer is $> R$, when the link from S to this peer, and the two links between the other two peers are removed.

The oblivious algorithm is a variation of the algorithm in our technical report [25]. Similar to the algorithm in Section 8, this oblivious algorithm operates in rounds and have four modes (the modes are slightly different from the ones in this paper). It uses the same coding strategy and schedules as round 1 and 2 in mode I in the algorithm presented in Section 8. Condition SC3 together with Equation 3 guarantee every pair of fault-free peers share at least R packets directly. When a peer, say A, detects an attack, instead of entering broadcast phase, A sends $(R - SA)^+$ packets received from B, which are inconsistent with packets from C, to S in round 3. SC3 guarantees that link AS has enough capacity. Then S compares these $(R - SA)^+$ packets with the ones it sends to B, and sends 1-bit notification to A stating B is faulty or good (implying C is faulty). When no attack is detected, A just sends any $(R - SA)^+$ packets received from B. The peers exchange the assessment at the end of round 3 as in the earlier algorithms and switch modes accordingly. And according to SC4, every peer receives at least R packets from the other two peers in round 2. So in mode III, the accused peer can recover the data from the packets it receives in round 2 using the same schedule, as long as the other two peers do not detect an attack.

13 General Networks

We have already seen that our algorithms achieves a 100% improvement in agreement throughput compared with the traditional Byzantine agreement algorithms for the four node network in Figure 1(a). For larger networks, a higher improvement can be achieved. In the five node complete network in Figure 5(b), every directed link has capacity R in both directions. Again, the traditional Byzantine agreement algorithms can achieve throughput R bits/unit time. Round 1 schedule for a modification of our algorithm for this particular five node network is illustrated in Figure 5(b), in which node S sends packet a to A, b to B, c to C and $a \oplus b \oplus c$ to D. The size of the packets is R bits each. This algorithm can achieve agreement throughput approaching $3R$ bits/unit time.

Notice that the necessary conditions we discussed in the previous sections are tight for this example 5-node network. However, in general, the necessary conditions are *not tight* for larger networks. For instance, consider the *complete* seven node network shown in Figure 5(c). Each directed link shown in the figure has capacity R in each direction, while the links not shown in the figure all have capacity ϵ such that $R >> \epsilon \approx 0$. We can see that necessary conditions NC1 and NC2 are both satisfied, however, it is not possible to achieve agreement at rates close to R.

14 Secure Multicast with Feedback

Inspired by the Byzantine agreement algorithms described in previous sections, we propose a secure multicast algorithm for a class of network topologies. This secure multicast algorithm achieves throughput for error correction arbitrarily close to the capacity in the presence of single node failure at the intermediate nodes, while only error detection and feedback are performed in the normal operations.

Consider a network $G(V, E)$ with V as the set of nodes and E as the set of links. There is one multicast session with one source $S \in V$, and a set of K destinations $D = \{D_1, \ldots, D_K\} \subset V$. The source and destinations are assumed to be fault-free, and at most one of the other nodes may fail. The proposed algorithm requires that the source S can communicate with each destination D_i *reliably* at some rate > 0 in both directions, and each $v \in V \backslash D \backslash \{S\}$ can communicate with $S \bigcup D$ and D with some rate > 0 in both directions. In particular, we consider the network topologies $G(V, E)$ that satisfy the following two constraints:

- There is a direct link from S to D_i, or there are at least three node-disjoint paths (to allow single fault tolerant) from S to D_i for all $i = 1, \ldots, K$, and vice versa.
- For every $v \in V \backslash D \backslash \{S\}$, there is a direct link from a node in $S \bigcup D$ to v, or there are at least three node-disjoint paths from $S \bigcup D$ to v (For example: S has three node-disjoint paths to v, or S, D_1 and D_2 each has one path to v and they are node-disjoint), and vice versa.

Let us assume that C is the capacity for $G(V, E)$ in spite of a single node failure at any one of $V \backslash D \backslash \{S\}$. In other words, for all $R < C$, there exists a network code such that every destination can recover the correct data at rate R no matter what the adversary node does. Since fault tolerant is possible at rate R, failure detection must be possible too. So there must be a network code NC such that misbehavior by a faulty node $v \in V \backslash \{D, S\}$ will either be detected by at least one destination, or all destinations will recover the correct data correctly at rate R. Then the algorithm is carried out as follows:

Procedure:

1. Node S transmits data to the destinations D according to the error detecting code NC at rate R.

2. Each destination node D_i tries to decode the received data. In acknowledgement of reception, D_i sends a bit '0' to S if no failure is detected; otherwise D_i sends '1'. This acknowledgement is transmitted reliably, either on the direct link D_iS, or through three node-disjoint paths from D_i to S.

3. The source S broadcasts the K acknowledgement bits from D to all nodes in the network, either on the direct link from S, or through three node-disjoint paths from S. If all K acknowledgement bit are '0', goto Step 1 and transmit a new generation of data. If any acknowledgement bit received by S is '1', the broadcast phase is entered:

   (a) Every intermediate node $v \in V \backslash D \backslash \{S\}$ sends all the data it has sent/received in Step 1 to $S \bigcup D$, either on a direct link, or through three node-disjoint paths as specified in the second constraint.

   (b) Each destination node D_i sends all the data it has sent/received in Steps 1 and 3a to the source S reliably, either on the direct link, or through three node-disjoint paths as specified in the first constraint.

   (c) Since there is at most one misbehaving node, Steps 3a and 3b together ensure that S can correctly retrieve exactly what each node v claims it has sent/received in Step 1, using a majority vote, similar to the traditional Byzantine agreement algorithm. Based on the information received during the broadcast phase, S determines the set of *suspected* links E', such that for each link $e \in E'$, what $e.tail$ claims it has sent on e is inconsistent with what $e.head$ claims it has received on e. Similar to the discussion on the diagnosis graph in Section 7, a link e is in E' only if one of the two end-points of e is faulty.

   Moreover, S tries to identify the faulty node: (1) If $|E'| \geq 2$, the links in E' must be all associated with a common node, and this common node is the faulty node; (2) If for a node v, what v claims to have sent is inconsistent with what it claims to have received in Step 1, v must be faulty; (3) If a suspected link is associated with S or one destination D_i, the other node associated with this suspected link must be the faulty one. It should be not hard to see that if $|E'| = 0$, node S must be able to identify the faulty node, otherwise the faulty node has sent data consistent with what it has received, in which case it is actually not misbehaving.

4. If S cannot identify the faulty node, $|E'| = 1$ and S finds a new error detecting code NC' of rate R for the network $G'(V, E \backslash E')$ with E' removed. Notice that throughput of rate R is still possible in G', since removing the suspected link is equivalent to the faulty node not sending anything on this link in the original network $G(V, E)$, and capacity of G' must be no smaller than C. So the new error detecting code NC' for G' must exist. Then S sends the new network code NC' and E' to every intermediate node reliably.

   Every node updates NC=NC' and goto Step 1. With NC', all destinations will be able to recover the correct data at rate R unless the faulty node misbehaves on its outgoing link(s) other than E'. Once it misbehaves on these outgoing link(s), the attack will be detected by at least one destination. Then the broadcast phase is entered again, and the faulty node will be identified.

5. If the faulty node is identified as node v, S finds a new network code NC" of rate R for the network $G''(V \backslash \{v\}, E)$ with the faulty node removed. The new network code NC" must exist, since removing the faulty node is equivalent to the faulty node v misbehaves by sending nothing in the original network $G(V, E)$, and the destinations must be still able to recover the correct data at rate R. The source S then broadcast NC" to all nodes reliably. The future data is transmitted according to NC" and no more broadcast phase will be performed since the faulty node has been identified.

Similar to BAC and BAO, in this multicast algorithm, this multicast algorithm is executed in generations in a pipelined fashion. In the normal operations, in which no failure is detected, the dissemination of the acknowledgement bits costs a constant overhead, whose impact on throughput can be made arbitrarily close to zero by increasing the duration of the clock cycles. The broadcast phase is performed at most twice throughout the execution of the multicast algorithm, so the average overhead *per cycle* of the broadcast phase decreases to zero as time increases. Thus, by using a suitably large clock cycle, the throughput can be made arbitrarily close to R. Hence this algorithm can achieve multicast throughput arbitrarily close to capacity C in spite of a single failure.

The multicast algorithm can easily be extended to tolerate multiple failures, which higher connectivity requirement. With single failure, the broadcast phase is performed for at most twice. The number of times the broadcast phase is performed increases as the number of faulty nodes increases.

15 Discussion

The definitions of throughput and capacity in this paper can be extended to other forms of agreement or consensus as well. For instance, not all nodes in the network may be peers. That is, the network may include nodes that are neither sender nor peers. Such nodes may assist in achieving agreement. *Consensus* is often defined as agreeing on an identical value as a function of the values being proposed by all the peers (so, all the peers act as "senders"). In particular, if all the fault-free peers propose an identical value, then the agreed value should be this particular value. It should be easy to see that the definition of throughput in this paper can be extended to this scenario as well. It should also be easy to see that the actual throughput for the different versions of consensus or agreement may be quite different.

The above definitions can be extended to the more general problem of *function computation*, where the agreed value may be an arbitrary function of the values proposed by the various nodes, and also to the notion of approximate agreement (in case of *approximate agreement*, one can trade-off throughput with "accuracy" of agreement).

All of these interesting problems are currently open. In this report, we consider the simplest instance of the agreement problem in a four-node network, as the first step towards addressing these problems.

16 Conclusions

In this paper, we define throughput and capacity of Byzantine agreement. We introduce necessary conditions that serve to define an upper bound on the agreement throughput. We prove the tightness of these conditions for the complete four-node network by constructing a capacity-achieving algorithm BAC. We also prove the tightness of four conditions for four-node network with fewer than two uplinks, by constructing a capacity-achieving algorithm BAO. Inspired by the Byzantine agreement algorithms, we propose a secure multicast algorithm. Many problems remain open, including capacity characterization for larger topologies, and with larger number of failures.

Acknowledgements

We thank Jennifer Welch and Pramod Viswanath for useful discussions.

References

[1] H. Attiya and J. Welch. *Distributed Computing.* McGraw-Hill, 1998.

[2] N. Cai and R. W. Yeung. Network error correction, part ii: Lower bounds. *Communications in Information and Systems*, 6(1):37–54, 2006.

[3] M. Castro and B. Liskov. Practical byzantine fault tolerance and proactive recovery. *ACM Transactions on Computer Systems*, 2002.

[4] T. D. Chandra, R. Griesemer, and J. Redstone. Paxos made live: an engineering perspective. In *PODC '07*, pages 398–407, New York, NY, USA, 2007. ACM.

[5] E. C. Cooper. Replicated distributed programs. In *SOSP'85*, 1985.

[6] C. Dwork and Y. Moses. Knowledge and common knowledge in a byzantine environment: crash failures. *Inf. Comput.*, 88(2):156–186, 1990.

[7] C. Fragouli, D. Lun, M. Medard, and P. Pakzad. On feedback for network coding. In *Proc. of 2007 Conference on Information Sciences and Systems (CISS)*, 2007.

[8] R. Friedman, A. Mostefaoui, S. Rajsbaum, and M. Raynal. Distributed agreement and its relation with error-correcting codes. In *Proc. 16th Int. Conf. Distributed Computing*, 2002.

[9] T. Ho, B. Leong, R. Koetter, M. Medard, M. Effros, and D. Karger. Byzantine modification detection in multicast networks using randomized network coding, 2004.

[10] S. Jaggi, M. Langberg, S. Katti, T. Ho, D. Katabi, and M. Medard. Resilient network coding in the presence of byzantine adversaries. In *INFOCOM'07*, 2007.

[11] S. Kim, T. Ho, M. Effros, and S. Avestimehr. New results on network error correction: capacities and upper bounds. In *ITA'10*, 2010.

[12] R. Koetter and M. Medard. An algebraic approach to network coding. In *Information Theory, 2001. Proceedings. 2001 IEEE International Symposium on*, pages 104–, 2001.

[13] M. N. Krohn. On-the-fly verification of rateless erasure codes for efficient content distribution. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 226–240, 2004.

[14] L. Lamport and K. Marzullo. The part-time parliament. *ACM Transactions on Computer Systems*, 16:133–169, 1998.

[15] L. Lamport, R. Shostak, and M. Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4:382–401, 1982.

[16] S.-Y. Li, R. Yeung, and N. Cai. Linear network coding. *Information Theory, IEEE Transactions on*, 49(2):371–381, Feb. 2003.

[17] G. Liang and N. Vaidya. Capacity of byzantine agreement: Tight bound for the four node network. *Technical Report, CSL, UIUC*, February 2010.

[18] N. A. Lynch. *Distributed algorithms*. Morgan Kaufmann Publishers, 1995.

[19] T. Mizrahi and Y. Moses. Continuous consensus via common knowledge. In *TARK'05*, 2005.

[20] T. Mizrahi and Y. Moses. Continuous consensus with ambiguous failures. *Distributed Computing and Networking (Lecture Notes in Computer Science)*, 4904/2008:73–85, 2008.

[21] T. Mizrahi and Y. Moses. Continuous consensus with failures and recoveries. In *DISC '08*, 2008.

[22] M. Pease, R. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *JOURNAL OF THE ACM*, 27:228–234, 1980.

[23] F. P. Preparata, G. Metze, and R. T. Chien. On the connection assignment problem of diagnosable systems. *IEEE Trans. Electr. Comput.*, EC-16(6):848–854, Dec. 1967.

[24] M. O. Rabin. Efficient dispersal of information for security, load balancing, and fault tolerance. *J. ACM*, 36(2):335–348, 1989.

[25] N. Vaidya and G. Liang. Capacity of byzantine agreement (preliminary draft - work in progress). *Technical Report, CSL, UIUC*, January 2010.

[26] Z. Yu, Y. Wei, B. Ramkumar, and Y. Guan. An efficient signature-based scheme for securing network coding against pollution attacks. *INFOCOM 2008*, pages 1409–1417, April 2008.

A Appendix

A.1 Proof of Theorem 3

In this paper, we only consider deterministic algorithms.

Theorem 3 *To achieve an agreement on σ bits, at least 3 rounds of message exchange is necessary, if less than σ bits are sent on any incoming link at any peer.*

Proof: Suppose in contrary that agreement on σ bits can be achieved within 2 rounds, when less than σ bits are sent on a incoming link of a peer.

First consider the case when less than σ bits are sent on link SA. Notice that the messages S sends to the peers in the second round (if there are any) only depend on the value v that S wants to send, since S receives no feedback from the peers when round 2 begins. So we can express the two messages S sends to a peer X as one function of v: $M_{SX}(v)$. The message node X sends to Y in round 2 is a function of what X receives in round 1, so it is also a function of v, denote it as $M_{XY}(v)$. Now consider the following scenarios:

- Scenario 1: S is fault-free and wants to send v, B is faulty and sends $M_{BA}(v')$ to A in round 2. So by the end of round 2, A has received $M_{SA}(v), M_{BA}(v'), M_{CB}(v)$ from S, B and C.
- Scenario 2: S is fault-free and wants to send v', C is faulty and sends $M_{CA}(v)$ to A in round 2. So by the end of round 2, A has received $M_{SA}(v'), M_{BA}(v'), M_{CB}(v)$ from S, B and C.

In scenario 1, A must decide on v since B is faulty; while in scenario 2, A must decide on v' since C is faulty. Since the less than σ bits are sent on link SA, i.e. $|M_{SA}| < \sigma$, there must be two different values $v \neq v'$ of σ bits such that $M_{SA}(v) = M_{SA}(v')$. Thus A sees identical messages in both scenarios. As a result, in any deterministic algorithm , A should decide on the same vector in both scenarios, which leads to a contradiction.

Now consider the case when less than σ bits are sent on a link between peers, say link BA. Then there must be two different values $v \neq v'$ of σ bits such that $M_{BA}(v) = M_{BA}(v')$. Consider the following two scenarios:

- Scenario 1: S is faulty and sends $M_{SA}(v)$ to A, $M_{SB}(v)$ to B, and $M_{SC}(v')$ to C. So by the end of round 2, A has received $M_{SA}(v), M_{BA}(v), M_{CA}(v')$ from S, B and C.
- Scenario 2: S is faulty and sends $M_{SA}(v)$ to A, $M_{SB}(v')$ to B, and $M_{SC}(v')$ to C. So by the end of round 2, A has received $M_{SA}(v), M_{BA}(v'), M_{CA}(v')$ from S, B and C.

Scenario 1 is equivalent to the case when S wants to send v and C is faulty by sending $M_{CA}(v')$ to A, so A must decide on v. And scenario 2 looks the same for B and C as S wants to send v' and A is faulty, so B and C decide on v' and hence A must decide on v' too. However, since $M_{BA}(v) = M_{BA}(v')$, A has received identical information by the end of round 2. So A must decide on the same value in both scenarios, which leads to a contradiction.

□

Informally speaking, when the algorithm operating in a pipelined manner, each generation uses each link for some c amount of time. Suppose in each generation $Rc - f(c)$ bits are agreed on. If an incoming link at a peer has capacity $< R$, say $R - \delta$ for some constant $\delta > 0$, then in each generation, only $(R - \delta)c$ bits can be sent on that link. If R is achieved by making $c \to \infty$, $f(c)$ must be of the order of $o(c)$. Then there must exist some constant C such that $f(c) < \delta c$ for all $c \geq C$. Thus if $c \geq C$,

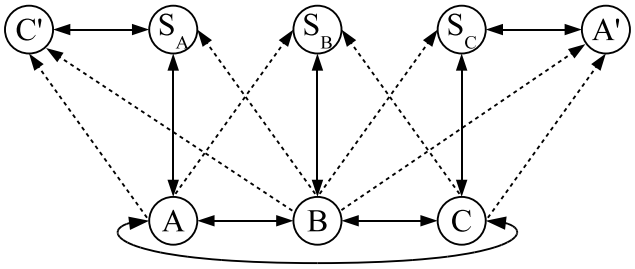


Figure 6: State machine for Theorem 4

then $Rc - f(c) > (R-\delta)c$, and by Theorem 3, at least three rounds of message exchange per generation is necessary to achieve agreement at rate R.

On the other hand, if R is achieved by using some $c < C$, then $f(c)$ must be $< \delta c$. Otherwise $f(c) \geq \delta c$, no more than $(R-\delta)c$ bits can be agreed on in each generation. Then the agreement throughput is at most $R-\delta$ and can never approach R. However, if $f(c) < \delta c$, then $Rc - f(c) > (R-\delta)c$, and again by Theorem 3, at least three rounds of message exchange per generation is necessary.

A.2 Proof of Theorem 4

Theorem 4 *To achieve agreement on σ bits, at least 4 rounds of message exchange is necessary, if less than σ bits are communicated through links between any one pair of peers.*

Proof: To prove this theorem, we construct the state machine as shown in Figure 6. S_A, S_B, and S_C run the exact same code as node S in the four node network. A and A' run the same code as node A in the four node network; B runs the same code as node B in the four node network; and C and C' run the same code as node C in the four node network. S_A, S_B, and S_C are initialized with the values v_A, v_B and v_C as the values to send. When node S is fault-free, v_A, v_B and v_C are identical. The uplinks from the peers to the "sources" are broadcast links. In particular, the messages A sends S_A and S_B are identical; the messages B sends to S_A, S_B, and S_C are identical; and the messages C sends to S_B and S_C are identical. Similarly, the messages B sends to A and A' are identical, the messages to C and C' are identical. For this proof, we will only consider scenarios when either S or B is faulty. Notice that when S is fault-free and B is faulty, i.e. $v_A = v_B = v_C$, nodes S_A, S_B, and S_C behave identically to a fault-free node S.

Suppose that agreement on σ bits can be achieved within 3 rounds in the four node network when less than σ bits are communicated through links AC and CA. Notice that the feedback S gets from the peers in round 2 only depends on the message the peers receive from S in round 1, so the message S sends to a peer X in round 3 can be expressed as $M'_{SX}(v, v_A, v_B, v_C)$. Similarly, the message a peer X sends to Y in round 3 can be expressed as $M'_{XY}(v_A, v_B, v_C)$. Consider the following three scenarios:

- Scenario 1: S wants to send v and B is faulty. B claims that S sends nothing to it, and claims that C sends $M_{CB}(v')$. Denote $\perp$ as the NULL message. Then this scenario is equivalent to the state machine when $v_A = v_B = v_C = v$ and B *pretends* that $v_B = \perp$ and $v_C = v'$. So by the end of round 3, A has received $[M_{SA}(v), M'_{SA}(v, v, \perp, v)]$, $[M_{BA}(\perp), M'_{BA}(v, \perp, v')]$, $[M_{CA}(v), M'_{CA}(M_{AC}(v), \perp, v)]$ from S, B and C, respectively.

- Scenario 2: S wants to send v' and B is faulty. B claims that S sends nothing to it, and claims that A sends $M_{AB}(v)$. This scenario is equivalent to the state machine when $v_A = v_B = v_C = v'$ and B *pretends* that $v_A = v$ and $v_B = \perp$. Similar to scenario 1, by the end of round 3 C has received

$[M_{SC}(v'), M'_{SC}(v', v', \perp, v')], [M_{AC}(v'), M'_{AC}(v', \perp, M_{CA}(v'))], [M_{BC}(\perp), M'_{BC}(v, \perp, v')]$ from S, A and B, respectively.

- Scenario 3: S is faulty. To A, S pretends that it wants to send v but B ignores its packets, and it also pretends the messages it receives from C are the ones C' sends to S_A in the state machine. To B, S sends nothing. To C, S pretends that it wants to send v' but B ignores its packets, and it also pretends the messages it receives from A are the ones A' sends to S_C in the state machine. This is equivalent to the state machine when $v_A = v$, $v_B = \perp$, and $v_C = v'$. We can show that by the end of round 3, it is possible that A has received $[M_{SA}(v), M'_{SA}(v, v, \perp, v)], [M_{BA}(\perp), M'_{BA}(v, \perp, v')], [M_{CA}(v'), M'_{CA}(M_{AC}(v), \perp, v')]$ from S, B and C, respectively; and C has received $[M_{SC}(v'), M'_{SC}(v', v', \perp, v')], [M_{AC}(v), M'_{AC}(v, \perp, M_{CA}(v'))], [M_{BC}(\perp), M'_{BC}(v, \perp, v')]$ from S, A and B, respectively.

We use M_{AC} and M_{CA} within M'_{CA} and M'_{AC} above in the following argument:

Since A and C can communicate through $M_{AC}, M'_{AC}, M_{CA}, M'_{CA}$ less than σ bits, there must be two different $v \neq v'$ of size σ bits such that

$$M_{AC}(v) = M_{AC}(v'); \tag{6}$$
$$M_{CA}(v) = M_{CA}(v'); \tag{7}$$
$$M'_{AC}(v, \perp, M_{CA}(v)) = M'_{AC}(v', \perp, M_{CA}(v')); \quad \textit{and} \tag{8}$$
$$M'_{CA}(M_{AC}(v), \perp, v) = M'_{CA}(M_{AC}(v'), \perp, v'). \tag{9}$$

By plugging the first two equations into the last two equations, we have

$$M'_{AC}(v, \perp, M_{CA}(v')) = M'_{AC}(v', \perp, M_{CA}(v')) \tag{10}$$
$$M'_{CA}(M_{AC}(v), \perp, v) = M'_{CA}(M_{AC}(v), \perp, v'). \tag{11}$$

Notice that A receives identical information in scenarios 1 and 3; C receives identical information in scenarios 2 and 3. So by the end of round 3, A cannot tell the difference between scenario 1 and 3, hence it must decide on v. Similarly, C cannot tell the difference between scenario 2 and 3, hence it must decide on v'. Then by round 3 in scenario 3, A decides on v and C decides on v', contradicts with the assumption. □

By an argument similar to the one in Appendix A.1, in achieving agreement capacity R, almost all generations have at least four rounds of message exchange if the sum capacity of the two links between a pair of peers is $R - \delta$ for some constant $\delta > 0$.

A.3 Optimizing our Algorithm

Although each generation of our algorithm consists of 3 rounds (except for the extended round 3), there are in fact at least four rounds of message exchanges in most cases:

- Mode I and II: one round of data transmissions in round 1, one round of data transmissions in round 2, and two rounds of notification dissemination in round 3 (one round to send out a peer's own notification, a second round to relay the notification from the other peers).

- Mode III when the sum capacity of the links between the two good peers is at least R: one round of data transmissions in round 1, one round of data transmissions between the two good peers in round 2, one round of data transmissions from the good peers to the accused peer, and two rounds of notification dissemination.

- Mode III when the sum capacity of the links between the two good peers is less than R: one round of data transmissions in round 1, one round of data transmissions between the two good peers and from the two good peers to the accused peer in round 2, one round of data transmissions between the accused peer and the two good peers in round 3, and two rounds of notification dissemination.

We can show that if a fault-free node does not detects an failure when finding the solution from the received packets and the notifications it receives in the first round of dissemination indicate no failure detected, the solution it finds must be the correct one. So we can optimize our algorithm as follows:

- Mode I and II: a fault-free node can decide after the first round of notification dissemination. Then it takes 3 rounds of message exchange to decide.
- Mode III when the sum capacity of the links between the two good peers is at least R: in round 3, the two good peers combine the data packets and their own notifications into one message. Then it takes only 3 rounds of message exchange to decide.
- Mode III when the sum capacity of the links between the two good peers is less than R: the fault-free nodes can decide after the first round of notification dissemination. Then it takes 4 rounds of message exchange to decide.
- Mode IV: there are only 2 rounds of data transmissions in mode IV.

So in the worst case, the faulty node misbehaves in a way such that the system stays in mode I, II or III, and it takes 3 to 4 rounds of message exchange for a fault-free peer to decide. According to Theorem 3 and Theorem 4, this algorithm is optimal in terms of number of rounds of message exchange.