Capacity of Byzantine Consensus in Capacity Limited Point-to-Point Networks

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

***Abstract*—In this paper, we investigate the problem of maximizing the *throughput*, i.e., achieving *capacity*, of Byzantine consensus in point-to-point networks, in which each link has a capacity constraint. We derive an upper bound of the capacity of consensus in general point-to-point networks, and prove its tightness in 4-node complete networks by construction. We also provide a probabilistically correct algorithm that achieves the upper bound in general networks.**

I. Introduction

In the last decade, we observe a tremendous growth in the popularity of data oriented online services, such as cloud computing, data centers and online storage. More and more enterprises run their critical business applications on data centers. Individual users also have become increasingly dependent on the Internet to store their personal data such as photos, musics, videos, etc. As the reliance of industry, government, and individuals on data centers and other similar online information services increases, the threat posed by malicious attacks and software errors has also become increasingly prominent. For example, software errors have brought down the Amazon S3 storage system for several hours (2008), and have caused well-known email services such as Gmail (2006) to wipe out customer's emails. Consequently, being able to provide reliable and consistent access to the data and services that they host has become the most basic and most important Quality-of-Service requirement that these online services must fulfill.

Byzantine Fault-Tolerant, i.e., BFT, provides a powerful state machine replication approach for providing highly reliable and consistent services in spite of the presence of failures. In BFT state machine replication, $n \geq 3f + 1$ replicas collectively behave as one *fault-free* server, even if up to f replicas are faulty and deviate from the protocol, i.e., *misbehave*, in arbitrary (*Byzantine*) fashions.

Despite the huge effort that has been devoted to the design of efficient BFT algorithms, one key point has long been overlooked: every algorithm must be implemented on top of a underlying communication network, and its performance is strongly coupled with the available resources, in other words, constraints of the underlying network. Very little is known about the relationship between the performance/efficiency of BFT algorithms and the constraints of the underlying network, as well as the design of *optimal* BFT algorithms under network constraints. It turns out that, when the available resource of the network are considered, classic solutions may perform quite poorly, especially when the resources are distributed unevenly.

In this paper, we study the design of optimal BFT algorithms taking the network constraints into account. In particular, we study the **capacity** of the Byzantine *Consensus* problem in point-to-point network.

A. Byzantine Consensus

The Byzantine consensus problem was first introduced by Pease, Shostak and Lamport [18]. This problem considers n nodes, namely $P_1, ..., P_n$, of which at most $f < n/3$ nodes may be *faulty* and deviate from the algorithm in arbitrary fashion. Each node P_i is given an input value v_i, and they want to agree on a value v such that the following properties are satisfied:

- *Termination*: every fault-free P_i eventually decides on an output value v_i',
- *Consistency*: the output values of all fault-free nodes are equal, i.e., for every fault-free node P_i, $v_i' = v'$ for some v',
- *Validity*: if every fault-free P_i holds the same input $v_i = v$ for some v, then $v' = v$.

The faulty nodes can engage in any kind of deviations from the algorithm, including sending false messages, collusion, and crash failures.

B. Point-to-Point Networks

We assume a synchronous network modeled as a directed graph $G(V, E)$, where V is the set of n nodes and E is the set of directed links. Each directed link $e(i, j) = (P_i, P_j) \in E$ is associated with a capacity $c(i, j)$, which specifies the maximum amount of information that can be transmitted on that link per unit time. That is, for any period of duration t and a link $e(i, j)$, up to $tc(i, j)$ bits of information can be sent from node P_i to P_j. If the

978-1-4673-0298-2/12/$31.00

link $e(i, j)$ does not exist, then communication *from* node P_i to P_j is impossible. We assume that all link capacities are integers. Rational link capacities can be turned into integers by choosing a suitable time unit. For irrational link capacities, it can be approximated by integers with arbitrary accuracy by choosing a suitable time unit.

C. Capacity of Consensus

Our goal in this work is to characterize the optimal achievable *throughput* of consensus. When defining throughput, the input value v_i at each node P_i referred in the above definition of consensus is viewed as an infinite sequence of *information* bits.

At each node P_i, we view the output value v_i' as 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 a consensus algorithm begins execution at time 0. In a given execution of a consensus algorithm, suppose that by time t all the fault-free nodes have agreed upon bits 0 through $b(t)-1$, and at least one fault-free node has not yet agreed on bit number $b(t)$. Then, the consensus *throughput* is defined as $\lim_{t\to\infty} \frac{b(t)}{t}$. The capacity of consensus is defined as follows:

Capacity of consensus in a given network $G(V, E)$, denoted as $C(G)$, is defined as the supremum of all achievable consensus throughputs.

The main contribution of this paper are threefold: (1) We introduce the problem of characterizing the capacity of consensus under network constraints; (2) We establish an upper bound of the consensus capacity of general networks; and (3) We show the upper bound is tight for complete 4-node networks.

II. Related Work

There has been significant research on agreement in presence of *Byzantine* or *crash* failures, theory (e.g., [11], [16], [1]) and practice (e.g., [6], [4]) both. Except for our previous work on the throughput of Byzantine *broadcast* in point-to-point networks [13], perhaps closest to our context is the work on *continuous consensus* [17], *multi-Paxos* [10], and *multi-valued Byzantine agreement* [8] 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. 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 *capacities of the network links*. The past work on multi-valued Byzantine agreement [8], [14], [15] 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.

III. Main Results

A. Upper Bound of Consensus Capacity

We prove an upper bound for the consensus capacity of any network $G(V, E)$, as a function of the link capacities of G. For any subset of nodes $S \subset V$ such that $|S| \leq f$, denote $\Gamma_S = \{\gamma : \gamma \subset V \backslash S \text{ and } |\gamma| = n - |S| - f\}$, i.e., Γ_S is the set of the $\binom{n-|S|}{n-|S|-f}$ subsets of $n - |S| - f$ nodes that do not include nodes in S. For every $\gamma \in \Gamma_S$, denote $I_S(\gamma) = \sum_{P_i \in \gamma, P_j \in S} c(i, j)$ as the incoming capacity to the set of nodes in S from nodes in γ, and let $I_S^* = \min_{\gamma \in \Gamma_S} I_S(\gamma)$ be the minimum over all $\gamma \in \Gamma_S$. Then we prove (in Appendix A) the following upper bound of the consensus capacity for general networks:

Theorem 1: For any network $G(V, E)$, the capacity of consensus satisfies

$$C(G) \leq I^* = \min_{S \subset V, |S| \leq f} I_S^*. \tag{1}$$

For a 4-node network with at most one faulty node, Theorem 1 implies that $C(G)$ is upper bounded by the minimum of the sum capacity of any two incoming links to a node, i.e.,

$$C(G) \leq I^* = \min_{i,j,k} \{c(j, i) + c(k, i)\}. \tag{2}$$

B. Tight Bound for Complete 4-Node Networks

We show that the upper bound from Theorem 1 is tight for complete 4-node networks (4-node networks in which all directed links have capacity > 0) with at most one failure, by constructing an algorithm that achieves throughputs arbitrarily close to I^* specified in Eq. 2.

Although the proposed algorithm only solves the problem for a special case of small networks, we consider it a good start for a solution to the more general cases. The proposed algorithm is non-trivial and interesting by itself. There are many problems in distributed computing in which optimal solutions are only known for small cases. For example, there are well known optimal solutions for the 2-party set disjointness problem [20], but not for the problem with more than 2 parties [5]. Similarly, the multi-party equality problem has a known optimal solution for 2 nodes, but not for 3 nodes in general [12].

IV. Structure of the Algorithm

The basic idea of our 4-node algorithm is to perform consensus "in parts". In particular, the sequence of input bits are divided into segments of equal size, and a sub-algorithm is used to achieve consensus for each segment sequentially, whereby limiting the total number of times the faulty nodes can cause a segment to fail. We will

refer to each execution of the sub-algorithm as a "generation". Each generation proceeds in three phases: (1) Input matching, (2) Consistency checking, and (3) Fault diagnosis. The algorithm has the following structure:
Divide the inputs into generations, and do the following for each generation in sequence:

0) Each node encodes its input of the current generation, which is represented by a certain number of *data packets*. These packets from the current generation are coded using a Reed-Solomon code to obtain *coded packets*.
1) **Input matching:** The nodes exchange a certain number of coded packets with each other to check whether there exist 2 nodes that have matching (or identical) inputs. If enough pairs of nodes appear to have different inputs, it can be guaranteed that the fault-free nodes do not have identical inputs, and then they can decide on a default output and terminate. When a set of at least 2 nodes appear to have identical inputs, the nodes in this set serve as one *virtual* source node and multicast their input to the other nodes.
2) **Consistency checking:** After the multicast of the input matching phase, the nodes check for the consistency of the received coded packets according to the Reed-Solomon code being used (as elaborated in the next section). If some node finds its received packets inconsistent, failure (i.e., misbehavior by a faulty node) during the previous phase is detected.
   a) If no one detects a failure, then all fault-free nodes will decide on an identical output and consensus of the current generation is successful. Repeat the process for the next generation.
   b) If some node detects a failure, then the consensus of the current generation fails and the fault diagnosis phase is invoked.
3) **Fault diagnosis:** After failure is detected, every node broadcasts to all other nodes everything it has sent and received so far in the current generation. This broadcast is made reliable using an error-free 1-bit Byzantine broadcast algorithm that tolerates $f < n/3$ Byzantine failures [7], [3]. This 1-bit broadcast algorithm is referred as **BC_Bit** in our following discussion. From the broadcast content, the fault-free nodes learn some information about the potential identity of the faulty node. In particular, the location of the faulty node will be narrowed down to a set of at most 2 nodes. Then the current generation is repeated with modified input matching and consistency checking phases to incorporate this information.

Similar structure called "dispute control" has been used in the context of secure multiparty computation [9], [2]. We have also used this structure for Byzantine broadcast in point-to-point networks [13], as well as to reduce the communication complexity of unconstrained Byzantine consensus [14]. It turns out that broadcast differs from consensus substantially and their capacity is different too.

The main challenge here lies in designing the coding and multicasting strategies so that the throughput of consensus is maximized while the total usage of every link is kept within its capacity limit. For 4-node networks, we are able to do so by exploiting some specific structure of the problem. However, for larger network, even though the general algorithm structure can be potentially extended, designing optimal coding and multicasting strategies remains an open problem.

V. Capacity of Complete 4-Node Networks

In this section, we prove that the upper bound from Theorem 1 is tight for complete 4-node networks (4-node networks in which all directed links have capacity > 0) with at most 1 faulty node, by constructing an algorithm that achieves throughputs arbitrarily close to I^*.

Here we use notations slightly different from the previous sections. We rename the four nodes as A, B, C and D, instead of $P_1, \cdots, P_4$. We denote by XY the directed link from node X to node Y.

When it is clear from context, we use XY to represent the capacity of the directed link XY. We use the notation $\widehat{XY}$ to represent the pair of directed links XY and YX, as well as the sum capacity of this pair of links: $XY + YX$. With this notation, Eq. 2 can be rewritten as

$$C(G) \leq I^* = \min \{ \begin{array}{l} BA + CA, BA + DA, CA + DA, \\ AB + CB, AB + DB, CB + DB, \\ AC + BC, AC + DC, BC + DC, \\ AD + BD, AD + CD, BD + CD \end{array} \} \quad (3)$$

A. Coding structure and properties for input matching and consistency checking

As discussed in Section IV, the algorithm proceeds in generations. In particular, for any integer $R \leq I^*$, the input values are divided into generations of $R\alpha$ bits for some positive integer α (the choice of α will be discussed later). Denote by $X(g)$ the input value at node X in the g-th generation. We use Reed-Solomon codes for input matching and consistency checking (potentially other codes may be used instead in general). In particular, the $R\alpha$-bit input value $X(g)$ is represented by R *data packets*, each being a symbol from Galois Field GF(2^α). The *coded packets* are computed as linear combinations of the R data packets, such that every subset of R coded packets represent a set of linearly independent combinations of the R data packets. As known from the design of Reed-Solomon codes, if α is chosen large enough, this linear independence requirement can be satisfied. The weights or coefficients used to compute the linear combinations are a part of the algorithm specification, and are assumed to be correctly known to all nodes a priori.

List 1 Basic operations of the proposed algorithm

a) The pair of nodes (X,Y) compares x and y directly: We will use this operation only when $\widehat{XY} \geq R$. Nodes X and Y each generates XY and YX independent coded packets from x and y, respectively, and exchange these packets over links XY and YX. Then node X checks if the packets received on link YX are consistent with its own input value x. Node X uses **BC_Bit** to reliably broadcasts a 1-bit notification "=" or "≠" indicating whether the packets received from Y are consistent with x or not. Node Y performs similar checks. If both notifications from X and Y are "=", then we say $x \equiv y$; otherwise we say $x \neq y$.

b) The pair of nodes (X,Y) compare x and y through node Z: Node X generates XY and XZ independent coded packets from x, and sends them to nodes Y and Z through links XY and XZ, respectively. Similarly, node Y sends YX and YZ coded packets generated from y to nodes X and Z, respectively. Node Z forwards $\min\{XZ, ZY\}$ coded packets received from node X to node Y, and forwards $\min\{YZ, ZX\}$ packets received from node Y to node X. Node X checks then if all the received packets are consistent with x. It then uses **BC_Bit** to broadcast a 1-bit notification "=" or "≠" indicating whether the packets received from Y and Z are consistent with x or not. Similar for node Y. If both notifications from nodes X and Y are "=", we say $x \overset{Z}{\equiv} y$; otherwise we say $x \overset{Z}{\neq} y$.

c) Diagnose the system: If failure is detected in the g-th generation , every node broadcasts all the packets it has sent or received in the current generation, with **BC_Bit**. By comparing the broadcast information from different nodes, the fault-free nodes are able to either narrow down identity of the faulty node within a set of two nodes, or correctly identify the faulty node.

We say that a coded packet is *consistent* with $X(g)$ if it is a valid linear combination of the R data packets of $X(g)$ with the given coefficients. Then the linear independent property implies that any subset of R coded packets that are consistent with $X(g)$ can be used to reconstruct $X(g)$. This further means that, if two input values $X(g) = Y(g)$, then any coded packet generated from $X(g)$ must be consistent with $Y(g)$. The converse of this means that, if there is a coded packet which is consistent with $X(g)$ but inconsistent with $Y(g)$, then $X(g) \neq Y(g)$.

List 1 summarizes a number of operations that we will be using repeatedly in our upcoming discussion. Readers are advised to get familiar with them before going to the discussion of the algorithm.

B. The Proposed Algorithm for 4 Nodes

In designing our algorithm for 4-node networks, we will exploit of the property stated by the following theorem (Please see Appendix B for the proof).

Theorem 2: In the 4-node networks, for all $R \leq I^*$, there must be a subset {X,Y,Z}, such that $\widehat{XY} \geq R$ and $\widehat{XZ} \geq R$.

Without loss of generality, we assume that $\widehat{AB} \geq R$ and $\widehat{BC} \geq R$. Then the proposed consensus algorithm has six modes of operation, which are listed below. At time 0 (the first generation), the network starts in mode Undetected $A \equiv B \equiv C$.

- Undetected $A \equiv B \equiv C$: No failure has been detected yet, and it appears that $A(h) \equiv B(h) \equiv C(h)$ for all $h < g$.
- Undetected $A \equiv B \not\equiv C$: No failure has been detected yet, and it appears that $A(h) \equiv B(h)$ for all $h \leq g$ and $B(h) \not\equiv C(h)$ for some $h \leq g$.
- Undetected $A \not\equiv B \equiv C$: No failure has been detected yet, and it appears that $A(h) \not\equiv B(h)$ for some $h \leq g$ and $B(h) \equiv C(h)$ for all $h \leq g$.
- Undetected $A \not\equiv B \not\equiv C$: No failure has been detected yet, and it appears that $A(h) \not\equiv B(h)$ for some $h \leq g$ and $B(h) \not\equiv C(h)$ for some $h \leq g$.
- Detected: Failure has been detected, and the location of the faulty node has been narrowed down to a set of 2 nodes.
- Identified: The faulty node has been identified.

For the rest of this section, we focus our discussion on mode Undetected $A \equiv B \equiv C$.

1) Mode Undetected $A \equiv B \equiv C$: For the g-th generation, the algorithm operates in this mode if no failure has been detected so far, and $A(h) \equiv B(h) \equiv C(h)$ for all $h < g$. The first generation operates in this mode too. A generation in mode *Undetected* $A \equiv B \equiv C$ proceeds as described in Algorithm 2. The line numbers referred below correspond to the line numbers for the pseudo-code in Algorithm 2. Recall that we assume $\widehat{AB} \geq R$ and $\widehat{BC} \geq R$.

`Line 1:` In generation g, nodes A, B and C first encode $A(g)$, $B(g)$ and $C(g)$ represented by R data packets, into coded packets, respectively. Then they send AB, BA, BC and CB coded packets on links AB, BA, BC and CB, respectively. These $AB + BA + BC + CB$ coded packets are generated such that every subset of R such coded packets are linear independent combinations of the R data packets. Each of nodes A, B and C then checks if the received coded packets are consistent with its own input of the current generation. The results of these checking is then broadcast to all nodes reliably by using **BC_Bit**. Due to the use of **BC_Bit**, all fault-free nodes receive identical 1-bit notifications from nodes A, B and C. Using these notification, each node determines whether $A(g) \equiv B(g)$ or not, as well as whether $B(g) \equiv C(g)$ or not.
`Line 2:` It appears that $A(g) \not\equiv B(g) \not\equiv C(g)$. Since $A(g) \not\equiv B(g)$, then if nodes A and B are both fault-free, there must be a coded packet generated from $A(g)$ that is inconsistent with $B(g)$, or the other way around. As we have discussed at the beginning of this section, this means that $A(g) \neq B(g)$. So either (1) both nodes A and

Algorithm 2 Mode Undetected $A \equiv B \equiv C$ (generation g)

1) Nodes (A,B) *compare* $A(g)$ and $B(g)$ directly. Nodes (B,C) *compare* $B(g)$ and $C(g)$ directly. (Please see List 1 for definition of *compare directly*.)

There can be 4 outcomes:

2) If $A(g) \neq B(g)$ and $B(g) \neq C(g)$: Switch to mode Undetected $A \neq B \neq C$, and restart the current generation with the new mode.
3) If $A(g) \equiv B(g)$ and $B(g) \neq C(g)$: Switch to mode Undetected $A \equiv B \neq C$, and restart the current generation with the new mode.
4) If $A(g) \neq B(g)$ and $B(g) \equiv C(g)$: Switch to mode Undetected $A \neq B \equiv C$, and restart the current generation with the new mode.
5) If $A(g) \equiv B(g)$ and $B(g) \equiv C(g)$: The pair of nodes (A,C) compare $A(g)$ and $C(g)$ through node D. (Please see List 1 for definition of *compare through another node*.)
   a) If $A(g) \overset{D}{\neq} C(g)$: A failure is detected. Then we diagnose the system. The location of the faulty node will be either narrowed down to a subset of 2 nodes, or correctly identified by the fault-free nodes. Then the algorithm switches to mode Detected or Identified as defined previously for the following generations.
   b) If $A(g) \overset{D}{\equiv} C(g)$: Node B sends BD coded packets generated from $B(g)$ to node D on link BD. Node D checks if all subsets of R coded packets received from nodes A, B and C have identical unique solution, and broadcasts the outcome with **BC_Bit**.
      i) If all subsets of R coded packets have identical unique solution: Nodes A, B and C decide on the output of the current generation as $A(g)$, $B(g)$ and $C(g)$, respectively. Node D decides on the unique solution of the received packets.
      ii) Otherwise, failure is detected. Then we diagnose the system, and switch to mode Detected or Identified accordingly.

B are fault-free and have different inputs, or (2) one of these two nodes is faulty and *pretends* to have input different from the other's. Similar for the pair of nodes B and C. In this case, we switch to mode Undetected $A \neq B \neq C$, and repeat generation g in the new mode.

`Lines 3-4:` It appears that $A(g) \equiv B(g) \neq C(g)$ or $A(g) \neq B(g) \equiv C(g)$. We switch to modes Undetected $A \equiv B \neq C$ or Undetected $A \neq B \equiv C$, and repeat generation g in the new modes, respectively.

`Line 5:` It appears that $A(g) \equiv B(g) \equiv C(g)$. Nodes A and B have exchanged $AB + BA = \widehat{AB} \geq R$ coded packets and found them all consistent with $A(g)$ and $B(g)$. So if both nodes A and B are fault-free, by the property of the coding strategy, it follows that $A(g) = B(g)$. Likewise for $B(g)$ and $C(g)$. However, node B can be faulty and pretend to have the same input as the other two fault-free nodes. Thus, even though $A(g) \equiv B(g)$ and $B(g) \equiv C(g)$, if B is faulty, A and C may have different inputs. We need to perform some extra steps.

We require node A to generate AD coded packets from $A(g)$ and send them to node D on link AD. Then node D *relays* $\min\{AD, DC\}$ coded packets received from node A to node C on link DC. Similarly, node D relays $\min\{CD, DA\}$ coded packets from node C to node A. Also, nodes A and C exchange $AC + CA$ coded packets on links AC and CA directly. Note that the links to/from node D, and the pair of links connecting nodes A and C were not used in `Line 1`. In `Line 5`, nodes A and C have exchanged directly or through node D:

$$\begin{aligned} & AC + CA + \min\{AD, DC\} + \min\{CD, DA\} \\ = \; & \min\{AC + CA + (AD + CD), AC + AD + (CA + DA), \\ & (AC + DC) + CA + CD, AC + DC + (CA + DA)\} \\ \geq \; & R \text{ coded packets.} \end{aligned} \tag{4}$$

The last inequality is due to Eq.3 that I^* is no more than $(AD + CD)$, $(CA + DA)$ and $(AC + DC)$.

`Line 5(a):` $A(g) \overset{D}{\neq} C(g)$ means that at least one of the coded packets that node C received in `Line 5` is inconsistent with $C(g)$, or at least one of the packets that node A received in `Line 5` is inconsistent with $A(g)$. This implies that, if nodes A, C and D did not misbehave, then $A(g) \neq C(g)$. However, the condition that $A(g) \equiv B(g) \equiv C(g)$ means that if nodes A, B and C did not misbehave, then $A(g) = B(g) = C(g)$. So we have a contradiction, which implies that the faulty node has misbehaved in the current generation.

Then we enter the fault diagnosis phase, and require every node to broadcast its input of the current generation and all coded packets it has received or sent so far in the current generation, using **BC_Bit**. By comparing the broadcast information, the fault-free nodes will be able to narrow down the potential identity of the faulty node to a set of at most 2 nodes. Then the system will switch to modes Detected or Identified accordingly. The diagnosis process has a similar structure as the "dispute control" from [9], [2]. We include a detailed discussion in Appendix C.

`Line 5(b):` As we can see from the above discussion of `Line 5`, nodes A and C have exchanged directly or through node D in total at least R coded packets. So, if nodes A, C and D are all fault-free, $A(g) \overset{D}{\equiv} C(g)$ implies that $A(g) = C(g)$. Together with the condition that $A(g) \equiv B(g) \equiv C(g)$, it is not hard to see that no matter which node is faulty, it can be guaranteed that all the fault-free nodes in the set {A,B,C} (at least 2 such nodes) have identical inputs in the current generation. Nodes A

and C each has already sent AD and CD coded packets generated from $A(g)$ and $C(g)$ to node D in `Line 5`, respectively. Node B sends BD coded packets generated from $B(g)$ to node D in this step.

Then node D checks if all size-R subsets of coded packets received from nodes A, B and C have identical unique solution, and announces the outcome to the other nodes using **BC_Bit**. If yes (`Line 5(b)i`), node D sets the output to the unique solution of the received coded packets, and fault-free nodes in {A,B,C} set the output of the current generation equal to their inputs. Otherwise (`Line 5(b)ii`), the faulty node must have misbehaved. Then we diagnose the system and switch to modes Detected or Identified according to the outcome of diagnosis.

C. Correctness

The termination property is satisfied trivially. The following theorem shows that the consistency and validity properties of Algorithm 2 in mode Undetected $A \equiv B \equiv C$ are also satisfied:

Theorem 3: If the nodes decide on an output for the g-th generation in mode Undetected $A \equiv B \equiv C$, then the outputs are identical and equal to the input of this generation at the fault-free nodes in set {A,B,C}.

Proof: Observe that in mode Undetected $A \equiv B \equiv C$, the fault-free nodes can decide on an output only if $A(g) \equiv B(g)$, $B(g) \equiv C(g)$ and $A(g) \overset{D}{\equiv} B(g)$. As we can see from the previous discussion, the fault-free nodes (at least 2 such nodes) in the set {A,B,C} must have identical input values for the g-th generation. So in Step 5(b)i, the fault-free nodes in set {A,B,C} decide on identical outputs.

In addition, due to Eq. 3 node D receives $\geq R$ coded packets from each pair of nodes in the set {A,B,C}. Since at least two nodes in {A,B,C} are fault-free and have the same input for generation g, of which the unique solution is the input of the fault-free nodes in {A,B,C}, when node D does not detect a failure, its output value of the g-th generation equals to the input of the other fault-free nodes. ■

The operations in the other modes are similar, with the difference being in the manner in which the coding strategy is used. So the details of operations and proofs of other modes are omitted in the main text and are included in Appendix D.

D. Throughput Analysis

It is not hard to see that, in every generation, every link XY carries out transmissions of at most XY coded packets, and consensus is achieved for R data packets. Since the size of each coded packets is α bits, it follows that every link is used for at most α time units for transmissions of coded packets. So if we ignore the communication cost of other operations, the proposed algorithm achieves consensus on $R\alpha$ bits by using each link at most α time units, which follows that the throughput of consensus is R bits per time unit.

The throughput is reduced (slightly) by the overheads cost by the following operations: using **BC_Bit** for broadcasting 1-bit notifications, dropping existing generations when the algorithm switches between modes, and the fault diagnosis phases after failure is detected. Each execution of **BC_Bit** uses a constant number of bits on each link, independent of $R\alpha$ ($R\alpha$ is the number of bits agreed on in each generation). Since there are a constant number of executions of **BC_Bit** per generation, the reduction in throughput from performing **BC_Bit** can be made arbitrarily close to 0 by increasing α.

Additionally, in the worst case, the system will switch modes from Undetected $A \equiv B \equiv C$ to Undetected $A \equiv B \not\equiv C$ (or $A \not\equiv B \equiv C$), to Undetected $A \not\equiv B \not\equiv C$, to Detected and then to Identified. So there will be at most 4 mode transitions. Moreover, after the first fault diagnosis phase, the system will switch to either mode Detected or Identified. If it enters mode Detected, then after a second fault diagnosis phase, it enters mode Identified. So there are at most 2 fault diagnosis phases before the faulty node is identified. Hence, when we compute the consensus throughput over a large number of generations, according to the definition in Section I ($\lim_{t\to\infty} b(t)/t$), the reduction in throughput from mode transitions and performing fault diagnosis phases diminishes to 0 as t goes to ∞.

Now we can conclude that the proposed consensus algorithm can achieve any throughput $R < I^*$ in complete 4-node networks. Together with Theorem 1, this implies that $C(G) = I^*$ when G is a complete 4-node network.

E. Comparison with Existing Algorithms

In many consensus algorithms, such as [7], [3], every link must carry transmission of the whole input. As a result, the throughput of such algorithms is bounded from above by the minimum of link capacities. Consider a 4-node network in which all links have capacity $K \gg 0$ bits per unit time, except for link AB whose capacity is $\epsilon \approx 0$. In this example, traditional consensus algorithms can achieve throughput at most ϵ bits per unit time. On the other hand, as we have seen in the previous discussion, our consensus algorithm is able to achieve throughput arbitrarily close to $K + \epsilon$, which can be arbitrarily better than ϵ.

VI. Probabilistic Algorithm for Larger Networks

Although the algorithm presented in the previous section achieves the consensus capacity of 4-node networks, it is a difficult task to extend it to larger networks. The main obstacle lies in the design of a coding strategy for larger networks in presence of node failures that achieves the goals of input matching and consistency checking, while using each link within its capacity.

However, if we relax the requirement for consensus, by allowing a small probability of error, we are able to achieve throughput of consensus arbitrarily close to the upper bound specified by Theorem 1 for networks with arbitrary n and at most $f < n/3$ failures. In this section, we briefly discuss one of such *probabilistically correct* algorithms.

The probabilistic algorithm has a similar structure as we discussed in Section IV. The main difference is that it relies on a collision-resistant hash functions (for example [19]) for input matching and consistency checking. For any $R \leq I^*$, the probabilistic algorithm operates as follows (in the g-th generation):

1) Every node P_i picks a random key K_i, and compute a hash value $h_i = (K_i, H(v_i(g), K_i))$, where $v_i(g)$ is node P_i's input of the current generation, and $H()$ is a collision-resistant hash function. Then every node P_i broadcasts h_i reliably using **BC_Bit**.
2) Every node P_j checks the received hash values against its own input $v_j(g)$ (for each $i \neq j$, node P_j compares h_i with $(K_i, H(v_j(g), K_i)$.), and broadcasts a notifications with **BC_Bit**, each of which indicating whether or not h_i is consistent with $v_j(g)$.
3) Using the broadcast notifications, all nodes find an identical set of nodes, denoted as P_{match}, of maximum size such that for every pair of $P_i, P_j \in P_{match}$, P_i claims that $v_i(g)$ is consistent with h_j, and vice versa.
4) If $|P_{match}| < n - f$, it can be guaranteed that the fault-free nodes do not have identical inputs. Then all fault-free nodes decide on a default output and terminate.
5) If $|P_{match}| \geq n - f$, then with high probability, the fault-free nodes in P_{match} have identical inputs for the current generation. So we require all nodes in P_{match} to act as one single virtual node and multicast their input value ($R\alpha$ bits) to the other nodes. Since $R \leq I^*$, according to the definition of I^*, it can be shown that this can be done by using each link for α time units by using linear network coding.
6) Every node $P_i \in P_{match}$ sets $v'_i(g) = v_i(g)$, and each node $P_i \notin P_{match}$ sets $v'_i(g)$ as the outcome of the multicast from the previous step. P_i picks a random key K'_i and compute $h'_i = (K'_i, H(v'_i(g), K'_i)$. Then it broadcasts h'_i using **BC_Bit**.
7) Every node P_j in or out of P_{match} checks the received hash values against its own outcome $v'_j(g)$ in a similar way as in step 2, and broadcasts a notification with **BC_Bit**, indicating whether or not all hash values and keys are consistent with $v'_j(g)$.
8) If all nodes claim that their outputs consistent with all received hash values, then every node P_j decides on $v'_j(g)$. Otherwise failure is detected and we diagnose the system. After the diagnosis, a set of pairs of nodes will be identified, such that in every pair, at least one node is faulty. For the future generations, no communication will be performed between the two nodes of every such pair. Moreover, if a node appears in more than f such pairs, it must be faulty. Thus, it can be shown that after at most $f(f+1)$ diagnosis phases, all faulty nodes will be identified exactly [9], [14].

By choosing the length of the hash value sub-linear in $R\alpha$, and making α large enough, the reduction in throughput cost by the overhead of broadcasting the hash values can be made arbitrarily close to 0, and the probability of collision of the hash function $H()$ can be made arbitrarily small. So this approach achieves consensus with throughput arbitrarily close to I^*, with high probability.

VII. Conclusion

In this paper, we studied the capacity of Byzantine consensus under the constraints of a finite capacity of point-to-point links in the network. We identified upper bound on the achievable throughputs in general networks. Then we introduced a structure for the capacity achieving agreement algorithm in general networks. In addition, we presented capacity achieving algorithms for complete 4-node networks with arbitrary link capacity with at most 1 faulty node. Then we present a probabilistic correct algorithm that achieves the upper bound with high probability in general (larger) networks.

Acknowledgment

This research is supported in part by Army Research Office grant W-911-NF-0710287 and National Science Foundation award 1059540. 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.

References

[1] H. Attiya and J. Welch, *Distributed Computing*. McGraw-Hill, 1998.
[2] Z. Beerliova-Trubiniova and M. Hirt, "Efficient multi-party computation with dispute control," in *TCC*, 2006.
[3] P. Berman, J. A. Garay, and K. J. Perry, "Bit optimal distributed consensus," *Computer science: research and applications*, 1992.
[4] M. Castro and B. Liskov, "Practical byzantine fault tolerance," in *OSDI*, Berkeley, CA, USA, 1999, pp. 173–186.
[5] A. Chakrabarti, S. Khot, and X. Sun, "Near-optimal lower bounds on the multi-party communication complexity of set disjointness," in *IEEE CCC*, 2003.
[6] T. D. Chandra, R. Griesemer, and J. Redstone, "Paxos made live: an engineering perspective," in *PODC '07*. New York, NY, USA: ACM, 2007, pp. 398–407.
[7] B. A. Coan and J. L. Welch, "Modular construction of a byzantine agreement protocol with optimal message bit complexity," *Inf. Comput.*, vol. 97, no. 1, pp. 61–85, 1992.
[8] M. Fitzi and M. Hirt, "Optimally efficient multi-valued byzantine agreement," in *PODC '06*, 2006.
[9] M. Hirt, U. Maurer, and B. Przydatek, "Efficient secure multi-party computation," in *ASIACRYPT 2000*, 2000.
[10] L. Lamport and K. Marzullo, "The part-time parliament," *ACM Transactions on Computer Systems*, vol. 16, pp. 133–169, 1998.

[11] L. Lamport, R. Shostak, and M. Pease, "The byzantine generals problem," *ACM Trans. on Programming Languages and Systems*, 1982.
[12] G. Liang and N. Vaidya, "Multiparty equality function computation in networks with point-to-point links," in *SIROCCO*, 2010.
[13] ——, "Capacity of byzantine agreement with finite link capacity," in *INFOCOM 2011*, 2011.
[14] ——, "Error-free multi-valued consensus with byzantine failures," in *ACM PODC*, 2011.
[15] ——, "Experimental performance comparison of byzantine fault-tolerant protocols for data centers," in *IEEE INFOCOM*, 2012.
[16] N. A. Lynch, *Distributed algorithms*. Morgan Kaufmann, 1995.
[17] T. Mizrahi and Y. Moses, "Continuous consensus with failures and recoveries," in *DISC '08*, 2008.
[18] M. Pease, R. Shostak, and L. Lamport, "Reaching agreement in the presence of faults," *JOURNAL OF THE ACM*, 1980.
[19] M. N. Wegman and J. L. Carter, "New hash functions and their use in authentication and set equality," *Journal of Computer and System Sciences*, vol. 22, pp. 265 – 279, 1981.
[20] A. C.-C. Yao, "Some complexity questions related to distributive computing(preliminary report)," in *STOC*. New York, NY, USA: ACM, 1979, pp. 209–213.

Appendix A
Proof of Theorem 1

Theorem 1: For any network $G(V, E)$, the capacity of consensus satisfies

$$C_{con}(G) \leq I^* = \min_{S \subset V, |S| \leq f} I^*_S. \quad (5)$$

Proof: Suppose on contrary that $C_{con}(G) > I^*$. Then there must exist a consensus algorithm, say *ALG*, that achieves consensus on $b(t) > tI^*$ bits during some period $[0, t]$, with no more than $tc(i, j)$ bits transmitted over every link $e(i, j) \in E$.

Without loss of generality, assume that $I^* = I_S(\gamma)$ for $S = \{P_1, \cdots, P_{|S|}\}$, and $\gamma = \{P_{|S|+1}, \cdots, P_{n-f}\}$. Now consider the state machine $G'(V, E')$ illustrated in Figure 1. $G'(V, E')$ is a subgraph of $G(V, E)$ with links between nodes in S and nodes in set $F = \{P_{n-f+1}, \cdots, P_n\}$ removed. The remaining links have the same capacity as in $G(V, E)$. So nodes in S receive nothing from nodes in F, and vice versa nodes in F receive nothing from nodes in S. In this state machine, every node P_i runs the correct code that node it should run in $G(V, E)$ according to algorithm *ALG*. Nodes in S are all given an identical input value y of $b(t)$ bits, and all the other nodes are given the same input value x of $b(t)$ bits, as illustrated in Fig.1.

Now consider the following two scenarios in the original network $G(V, E)$:

1) All nodes in S are fault-free and given input value y, $P_{|S|+1}, \cdots, P_{n-f}$ are fault-free and given input value x. Only nodes in F are faulty. The f faulty nodes behave as in the state machine $G'(V, E')$ by not sending anything to nodes in S and ignoring any incoming information received from S.
2) $P_{|S|+1}, \cdots, P_n$ are fault-free and given input value x. Only the $|S| \leq f$ nodes in S are faulty and behave as in the state machine $G'(V, E')$ by not sending anything to nodes in F and ignoring any incoming information received from nodes in F. According to the validity property, the output value of $P_{|S|+1}, \cdots, P_n$ is x.

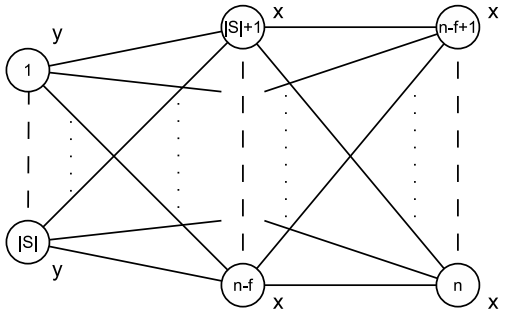

Fig. 1. State machine for the proof of Theorem 1.

Given that algorithm *ALG* solves consensus of $b(t)$ bits by time t, the nodes will reach consensus correct by time t in both scenarios. It is not hard to see that, in $G(V, E)$, the information observed by $P_{|S|+1}, \cdots, P_{n-f}$ in both scenarios is the same as the information observed in the state machine $G'(V, E')$. As a result, they cannot distinguish between the two scenarios and must decide on the same value. So in scenario 1, $P_{|S|+1}, \cdots, P_n$ decide on the output value x. Then according to the consistency property, nodes in S will also decide on the same output value x – the input value of $P_{|S|+1}, \cdots, P_{n-f}$.

Now let us fix y in scenario 1 and vary x. Notice that in scenario 1 nodes in S receive no more than tI^* bits from nodes in γ. Since we assume that $b(t) > tI^*$, according to the pigeonhole principle, there must be two values $x_1 \neq x_2$ such that the resulting tI^* bits received by nodes in S from nodes in γ are identical. As a result, nodes in S must decide on an identical output value, namely x^*, when $P_{|S|+1}, \cdots, P_{n-f}$ are given the input value x_1 and x_2. However, as we discussed above, output value of nodes in S equals to the input value of γ. So nodes in S must decide on two different output values when $P_{|S|+1}, \cdots, P_{n-f}$ are given two different input values x_1 and x_2, which leads to a contradiction. ■

Appendix B
Proof of Theorem 2

Proof: To prove this theorem, we first show that for any positive value $R \leq I^*$, at least three of the six pair of links, i.e., $\widehat{AB}, \widehat{AC}, \widehat{AD}, \widehat{BC}, \widehat{BD}, \widehat{CD}$ are $\geq R$. Consider any subset of three nodes, say $\{A, B, C\}$. We have

$$\begin{aligned} \widehat{AB} + \widehat{AC} + \widehat{BC} &= (BA + CA) + (AB + CB) + (AC + BC) \\ &\geq 3R. \end{aligned} \quad (6)$$

The inequality follows from Eq.3. It then follows that at least one of $\widehat{AB}, \widehat{AC}, \widehat{BC}$ is $\geq R$.

Without loss of generality, assume that $\widehat{AB} \geq R$. For the two subsets $\{A, C, D\}$ and $\{B, C, D\}$, according to the

same argument, at least one of $\{\widehat{AC}, \widehat{AD}, \widehat{CD}\}$ and one of $\{\widehat{BC}, \widehat{BD}, \widehat{CD}\}$ are $\geq R$. There are two cases:

- $\widehat{CD} < R$: It follows that one of $\{\widehat{AC}, \widehat{AD}\}$ and one of $\{\widehat{BC}, \widehat{BD}\}$ are $\geq R$. So at least three pairs of links that are $\geq R$.
- $\widehat{CD} \geq R$: In this case, we have

$$\widehat{AC} + \widehat{AD} + \widehat{BC} + \widehat{BD} = (CA + DA) + (CB + DB) + (AC + BC) + (AD + BD) \geq 4R. \quad (7)$$

So at least one of $\widehat{AC}, \widehat{AD}, \widehat{BC}, \widehat{BD}$ is $\geq R$. Again, we have three pairs of links that are $\geq R$.

Now we have shown that at least 3 pairs of links are $\geq R$. Then it is easy to see that at least two pairs of them are adjacent. Then the theorem follows. ■

Appendix C
Fault Diagnosis

In this section, we describe how fault diagnosis is done when failure is detected in `Line 5(a)` of Algorithm 2. In `Line 5(a)`, we have $A(g) \equiv B(g)$, $B(g) \equiv C(g)$ and $A(g) \overset{D}{\neq} C(g)$. To diagnose the system, we execute the following operations:

1) Every node broadcasts its input of the current generation and the coded packets it has sent or received so far in generation g, using **BC_Bit**. Due to the use of **BC_Bit**, all fault-free nodes obtain identical information of what every node claims to have sent and received.
2) For each packet transmitted in `Lines 1` and `5` Algorithm 2, each fault-free node 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 faulty node must be one of {X,Y}.
3) If more than one pairs is identified in step 2, since the claims from two fault-free nodes should never mismatch, all the identified pairs must contain one common node, which must be the faulty node. In this case, the faulty node is correctly identified.
4) If no pair of nodes are identified in step 2, which means that all claims from different nodes about the transmitted coded packets match, we investigate each node individually:
   a) Node A: If the broadcast $A(g)$ differs from the unique solution of the packets exchanged on links AB and BA, then node A must be faulty. Or if node A has broadcast "$\neq$" in `Line 5` but $A(g)$ equals to the unique solution of the packets exchanged on links AC, CA, AD and DA, then node A must be faulty.
   b) Node B: If the broadcast $B(g)$ differs from the unique solution of the packets exchanged on links AB and BA, then node B must be faulty. Similarly, if the broadcast $B(g)$ differs from the unique solution of the packets exchanged on links BC and CB, then node B must be faulty.
   c) Node C: Similar to node A.
   d) Node D: If the packets node D received on link AD mismatch with the ones node D sent on link DC, then node D must be faulty. Similarly, if the packets node D received on link CD mismatch with the ones node D sent on link DA, then node D must be faulty.

It can be verified that under the condition of $A(g) \equiv B(g)$, $B(g) \equiv C(g)$ and $A(g) \overset{D}{\neq} C(g)$, one of the cases listed in Steps 2-4(d) above must occur. So we can either narrow down the potential identity of the faulty node to a set of two nodes, or identify the faulty node exactly.

The fault diagnosis procedure when failure is detected in other places is similar.

Appendix D
Operations of Other Modes

A. Mode Undetected $A \equiv B \not\equiv C$

For the g-th generation, the algorithm operates in this mode if no failure has been detected, $A(h) \equiv B(h)$ for all $h \leq g$, and $B(h) \not\equiv C(h)$ for some $h \leq g$ This mode proceeds as follows:

1) The pair of nodes (A,B) compare $A(g)$ and $B(g)$ directly.
2) If $A(g) \not\equiv B(g)$: The algorithm aborts the current generation, switches to mode Undetected $A \not\equiv B \not\equiv C$, and restarts the current generation with the new mode. Following generations will also operate in the new mode.
3) If $A(g) \equiv B(g)$: The pair of nodes (A,B) compare $A(g)$ and $B(g)$ through node C. Notice that no more communication over links $\widehat{AB}$ is needed.
   a) If $A(g) \overset{C}{\not\equiv} B(g)$: It contradicts with the condition $A(g) \equiv B(g)$. So failure is detected. Then we diagnose the system, and switch to mode Detected or Identified accordingly.
   b) If $A(g) \overset{C}{\equiv} B(g)$: Node C checks if the packets received from nodes A and B has an unique solution, and broadcasts a notification of the outcome with **BC_Bit**. If there is no unique solution, the failure is detected and diagnosis is performed. Otherwise, denote $C'(g)$ as the unique solution. Then the pair of nodes (A,C) compare $A(g)$ and $C'(g)$ through node D. Similar as step 3 above, no more communication over links $\widehat{AC}$ is needed. The rest is the same as steps 5(a) to 5(b)ii in mode Undetected $A \equiv B \equiv C$, by substituting $C(g)$ with $C'(g)$.

Theorem 4 states the correctness of the above steps.

Theorem 4: If the nodes decide on an output for the g-th generation in mode Undetected $A \equiv B \not\equiv C$, then the

decided outputs are identical and equal to the input of this generation at the fault-free node(s) in set {A,B} .

Proof: In mode Undetected $A \equiv B \not\equiv C$, the nodes can decide on an output only if $A(g) \equiv B(g)$, $A(g) \overset{C}{\equiv} B(g)$ and $A(g) \overset{D}{\equiv} C'(g)$. Notice that the number of coded packets exchanged between nodes B and C is

$$BC + \min\{AC, CB\} = \min\{BC + AC, BC + CB\} \geq R. \quad (8)$$

The inequality is due to $BC+AC \geq R$ and $BC+CB \geq R$. So $B(g) = C'(g)$ if both nodes B and C are fault-free. Similar to the proof of Theorem 3, after nodes (A,C) compare $A(g)$ and $C'(g)$ through node D, it can be guaranteed that the values $A(g), B(g), C'(g)$ are identical at the fault-free nodes in set {A,B,C}. Then the rest follows the proof of Theorem 3. ■

B. Mode Undetected $A \not\equiv B \equiv C$

Similar to mode Undetected $A \equiv B \not\equiv C$ but with roles of nodes A and C being swapped.

C. Mode Undetected $A \not\equiv B \not\equiv C$

For the g-th generation, the algorithm operates in this mode if no failure has been detected, $A(h) \not\equiv B(h)$ for some $h \leq g$, and $B(h) \not\equiv C(h)$ for some $h \leq g$. Mode Undetected $A \not\equiv B \not\equiv C$ proceeds as follows:

1) The pair of nodes (A,C) compare $A(g)$ and $C(g)$ through node B and also through node D.
2) If $A(g) \overset{B}{\not\equiv} C(g)$ and $A(g) \overset{D}{\not\equiv} C(g)$: In this case, it can be guaranteed that the fault-free nodes must have different input values. So the algorithm can decide on a default value and terminate.
3) If $A(g) \overset{B}{\equiv} C(g)$ and $A(g) \overset{D}{\not\equiv} C(g)$; or $A(g) \overset{B}{\not\equiv} C(g)$ and $A(g) \overset{D}{\equiv} C(g)$: Failure is detected. Then we diagnose the system, and switch to mode Detected or Identified accordingly.
4) If $A(g) \overset{B}{\equiv} C(g)$ and $A(g) \overset{D}{\equiv} C(g)$: Node D forwards as many packets received from nodes A and C to node B on link DB. Then nodes B and D check whether all size-R subsets of the received coded packets have an identical unique solution, and broadcast the outcomes with **BC_Bit**.
    a) If both nodes B and D find an identical unique solution of the received coded packets: Nodes A and C decide on $A(g)$ and $C(g)$ respectively. Nodes B and D decide on the unique solution of the received packets.
    b) Otherwise, failure is detected. Then we diagnose the system, and switch to mode Detected or Identified accordingly.

Theorem 5 states the correctness of the above steps.

Theorem 5: If the nodes decide on an output for the g-th generation in mode Undetected $A \not\equiv B \not\equiv C$, then the decided outputs are identical and equal to the input of this generation at the fault-free node(s) in set {A,C} .

Proof: In mode Undetected $A \not\equiv B \not\equiv C$, the nodes can decide on an output only if $A(g) \overset{B}{\equiv} C(g)$ and $A(g) \overset{D}{\equiv} C(g)$. Similar to the proof of Theorem 4, when nodes (A,C) compare $A(g)$ and $C(g)$ through node B, node B exchanges $\geq R$ packets with each one of nodes A and C. If we denote $B'(g)$ as the unique solution of the packets node B has received, it follows $A(g) \equiv B'(g)$ and $B'(g) \equiv C(g)$. Then the rest follows the same proof of Theorem 3. ■

D. Mode Detected

The algorithm operates in this mode if failure has been detected and the location of the faulty node is narrowed down to a subset of two nodes. For mode Detected, assumptions of $\widehat{AB} \geq R$ and $\widehat{BC} \geq R$ are not used. So without loss of generality, we can assume that the faulty node has been narrowed down to the set $\{B, D\}$. It is worth noting that in this case, all fault-free nodes know that nodes A and C must be fault-free.

1) The pair of nodes (A,C) compare $A(g)$ and $C(g)$ through node B and also through node D.
2) If $A(g) \overset{B}{\not\equiv} C(g)$ and $A(g) \overset{D}{\not\equiv} C(g)$: fault-free nodes A and C must have different input values, since one of nodes B and C must be fault-free. So the algorithm can decide on a default value and terminate.
3) If $A(g) \overset{B}{\equiv} C(g)$ and $A(g) \overset{D}{\not\equiv} C(g)$; or $A(g) \overset{B}{\not\equiv} C(g)$ and $A(g) \overset{D}{\equiv} C(g)$: Failure is detected. Then the full-broadcast is performed to identify the faulty node. The system then switches to mode Identified.
4) If $A(g) \overset{B}{\equiv} C(g)$ and $A(g) \overset{D}{\equiv} C(g)$: Nodes A and C decide on $A(g)$ and $C(g)$ respectively. Nodes B and D decide on the unique solution of the packets received from nodes A and C.

The proof of correctness of this mode is trivial. So we do not include it in this paper.

E. Mode Identified

The algorithm operates in this mode if the identify of the faulty node has been established. Without loss of generality, assume that the faulty node has been identified as node B. Fault-free nodes A, C, and D know that node B is faulty and the other nodes are fault-free.

1) The pair of nodes (A,C) compare $A(g)$ and $C(g)$ through node D.
2) If $A(g) \overset{D}{\not\equiv} C(g)$: In this case, fault-free nodes A and C must have different input values. So the algorithm can decide on a default value and terminate.
3) If $A(g) \overset{D}{\equiv} C(g)$: Nodes A and C decide on $A(g)$ and $C(g)$ respectively. Node D decides on the unique solution of the packets received from nodes A and C.

The proof of correctness of this mode is trivial. So we do not include it in this paper.