A Clean Slate Design for Secure Wireless Ad-Hoc Networks - Part 2: Open Unsynchronized Networks

Jonathan Ponniah
Department of Electrical and Computer Engineering
Texas A&M

Yih-Chun Hu
Department of Electrical and Computer Engineering
University of Illinois at Urbana-Champaign

P. R Kumar
Department of Electrical and Computer Engineering
Texas A&M

***Abstract*—We build upon the clean-slate, holistic approach to the design of secure protocols for wireless ad-hoc networks proposed in part one. We consider the case when the nodes are not synchronized, but instead have local clocks that are relatively affine. In addition, the network is open in that nodes can enter at arbitrary times. To account for this new behavior, we make substantial revisions to the protocol in part one. We define a game between protocols for open, unsynchronized nodes and the strategies of adversarial nodes. We show that the same guarantees in part one also apply in this game: the protocol not only achieves the max-min utility, but the min-max utility as well. That is, there is a saddle point in the game, and furthermore, the adversarial nodes are effectively limited to either jamming or conforming with the protocol.**

I. Introduction

The focus of this paper is on secure wireless ad-hoc networks. Security in this context implies that the data transfer between legitimate nodes is immune to the subversive efforts of adversarial nodes. Wireless *ad-hoc* networks, as a defining feature, lack a centralized controller; the nodes themselves are responsible for discovering neighbors, assigning priorities, determining schedules for achieving these priorities, identifying potential routes, monitoring the performance of existing routes, and arriving at a consensus by the end of each of these operations.

The challenge in securing a network like the one just described, or complex systems in general, is in anticipating all of the ways a process can fail and the effects of the failure on the system as a whole. A common approach to this task is known as "defense-in-depth" and mirrors the way one would secure a castle; by deploying multiple defensive layers, each of which protects an "exposed variable" (an entry point to the system) in the event the preceding layer is breached [1].

This approach, however, suffers from an unavoidable drawback; the possibility that some clever, sophisticated attack goes overlooked and/or some undetected structural vulnerability transcends multiple defensive layers. As a result, the process amounts to an arms race between more sophisticated attacks and more complicated fixes, [2] being such an example; at no point can we definitively claim to have obtained a protocol that is immune to all possible attacks.

In part one and in [3], we propose a clean-slate, holistic approach to the security of wireless ad-hoc networks that enables a collection of distributed nodes infiltrated with attackers to form a functioning network. We define a game between protocols and adversarial strategies, in which the protocol p is chosen first, and the adversarial strategy q_p is chosen in response. We define a payoff for this game $J(p, q_p)$ that depends on a utility function $U(x)$ of a throughput vector x, and an execution (p, q_p), and describe a protocol that satisfies the following claims:

(G1) The protocol is near max-min optimal with respect to the payoff $J(p, q_p)$.

It would appear that this payoff is the best any protocol can achieve, since the protocol is always chosen before the adversarial strategy. In fact, we do better:

(G2) The protocol is near min-max optimal with respect to the payoff $J(p, q_p)$, where the minimization occurs over adversarial strategies that are effectively confined to jamming.

The claim (G2) implies the existence of a saddle-point in the game between protocols and adversarial strategies; the adversarial nodes gain no advantage from knowing the protocol a priori. These guarantees are made without characterizing the full scope of attacks that can be deployed by the adversarial nodes, but are instead contingent on an underlying set of model assumptions; if the assumptions are violated, the guarantees cease to be valid.

In this paper, we will further develop the approach proposed in part one, by removing a model assumption that simplified the protocol design; the assumption that the nodes in the wireless network are born or turn on simultaneously. Instead we assume that the nodes in this

This material is based upon work partially supported by NSF Contract CNS-1302182, AFOSR Contract FA9550-13-1-0008, and NSF Science & Technology Center Grant CCF-0939370.

network are born at arbitrary times, and show that the guarantees made (G1)-(G2) still hold.

Before moving forward, we will discuss the implications of allowing nodes to be born at arbitrary times, and the reasons why this general case deserves more attention. One reason, concerns clock synchronization. Whereas in part one we assume the local clocks are synchronized here we assume the clocks are unsynchronized and relatively affine. Since the protocol relies on synchronized clocks to ensure the legitimate nodes can operate according to a common schedule, we will have to consider the possibility that adversarial nodes may undermine the process of clock synchronization by behaving maliciously.

The other reasons concern the challenges of operating "open networks"; those networks able to absorb nodes or subnetworks of nodes that were born recently.

First, a subnetwork must be able to detect the existence of an adjacent node or subnetwork in a timely fashion, for every moment in which these subnetworks remain segregated detracts from a utility-optimal throughput. Since the legitimate nodes in our model are half-duplex (they cannot transmit and receive messages simultaneously), such timely detection is not guaranteed. Each node operates according to its own transmit/receive schedule, and this schedule may not be orthogonal to that of an adjacent node in a different subnetwork. We introduce the role of a "sentinel" wherein a node periodically completely ceases transmission for a prolonged period of time, to ensure that any adjacent subnetworks will be detected, but not so frequently that the sentinel activity significantly reduces the throughput.

A second challenge is managing the timely merge of a pair of adjacent subnetworks. This process requires a subnetwork to pause its operations, merge with the outlier, and readjust the schedule to support a utility-optimal throughput for the newly enlarged network, all of which amounts to significant additional overhead. Since each merge attempt reduces the effective throughput of the participants, a legitimate network might be instigated into making numerous spurious merge attempts with an adversarial partner.

The third problem is somewhat unusual; at near infinite time, the time-stamps are of near-infinite size. As a result, any new legitimate node can never know while listening to an ongoing transmission from a neighbor, whether to keep listening for more bits or conclude that an adversarial node is feeding it an unlimited stream of noise. Hence, the legitimate nodes must operate with packets of fixed size L. However, this solution introduces a vulnerability of its own; the need to restart the clock whenever 2^L time units have expired. The reuse of timing packets allows an adversarial node to replay messages from a legitimate node that were originally transmitted in a previous clock iteration. This attack, appropriately known as the replay attack, is impossible to stop.

Without further ado, we can conclude that arbitrary birth times pose significant challenges to the operation of wireless ad-hoc networks. This paper will be devoted to resolving them.

II. The Model Assumptions

There are n nodes, some of which are legitimate and the rest adversarial. The legitimate nodes are half-duplex and do not know which of the other nodes are legitimate or adversarial. On the other hand, the adversarial nodes know their identities and can communicate via back channels of infinite bandwidth.

Each node is given a set of instructions, known as a protocol p, intended to form a functioning network operating at a utility-optimal throughput, from the distributed collective. The legitimate nodes are obligated to follow the protocol exactly. The adversarial nodes, on the other hand, devise a response q_p designed to subvert the protocol p. At time $t = 0$ with respect to some global reference clock, the first legitimate node turns on and proceeds to execute p. Subsequently, at arbitrary times, the remaining nodes both legitimate and adversarial turn on and begin to execute p and q_p respectively. The protocol p and the adversarial response q_p interact over the entire operating lifetime to form an effective throughput vector $x := f(p, q_p)$ of all possible $n(n-1)$ source-destination pairs, where n is the number of nodes. Since the legitimate nodes are born at arbitrary times, we need to define the time interval over which the throughput x is evaluated. In this paper, we will concern ourselves with the throughput evaluated from the birth time of the last legitimate node. Each element of the throughput vector denotes the throughput of a source-destination pair. The function f, models the physical properties of the network and reflects the model assumptions that we will spell out shortly.

First, we will first have to re-introduce some concepts and definitions that were explained in [3]: A *modulation scheme* $m \in \mathcal{M}_i$ specifies a physical mode of transmission by node i. A legitimate modulation scheme is associated with a message and a corresponding rate $r(m)$. An adversarial node also has the option of choosing, in addition to the legitimate modulation schemes, a jamming scheme that emits random noise at some specified power level.

The vector of modulation schemes corresponding to each source in the set of $n(n-1)$ one-hop, source-destination pairs is called a *concurrent transmission vector* $c := \{m_1, \ldots, m_{n(n-1)}\}$.

A concurrent transmission vector is *feasible* by definition if all the messages transmitted by the sources are received without error at the respective destinations. There are two factors that determine whether or not a concurrent transmission vector is feasible. The first is the physical channel itself. The second is more subtle; the ability of the adversarial nodes to exploit the physical channel and/or the back channel of infinite bandwidth and relay or transmit messages that would otherwise be lost. We denote the set of feasible concurrent transmission vectors by $\mathcal{F}$.

A feasible concurrent transmission vector can be actively *disabled* if any of the corresponding messages are lost when the modulation schemes specified of adversarial nodes are substituted with other modulation schemes (such as jamming schemes). The set of concurrent transmission vectors that can be actively disabled, which we denote by $\mathcal{D}$, is independent of the protocol.

A feasible concurrent transmission vector can also be *passively* disabled if it is prevented from ever being deployed by the protocol due to the actions of the adversarial nodes. The type of actions we refer to are protocol specific, and refer to such strategies as disseminating false information or sabotaging clock synchronization. We now state the model assumptions.

(M1) The set of legitimate modulation schemes is finite.

It follows from (M1) that the set of concurrent transmission vectors is also finite. We denote this set by $\mathcal{C} := \{c_j, j = 1, \ldots, N\}$ where each element is indexed from $1, \ldots, N$.

(M2) The legitimate nodes are connected. More precisely, there exists a path connecting the legitimate nodes, where each edge in the path corresponds to a modulation scheme of positive rate, and the edge belongs to a concurrent transmission vector that cannot be actively disabled by adversarial nodes.

(M3) Each node is able to perfectly encrypt its messages, and encrypted messages cannot be forged or altered by an adversarial node. In addition, each node possesses an identity certificate provided by a trusted authority.

(M4) Each node is equipped with a local clock, and the local clocks of the legitimate nodes are unsynchronized and affine.

III. A Game Between Protocols and Adversarial Strategies

We now set up a zero-sum game between protocols and adversarial strategies. First we choose a protocol p and announce the protocol to all the nodes, both adversarial and legitimate. The adversarial nodes, after observing p, choose a strategy q_p in response. We will need to define the payoff $J(p, q_p)$ of the game, as a function of the effective throughput vector x, where x is evaluated over the interval $[T_l, T_l+T)$, and T_l is the birth time of the last legitimate node. The payoff is a measure of how effectively the protocol p can maintain data transfer between legitimate nodes when the adversarial nodes choose a strategy q_p.

Let $U(x)$ denote the utility accrued to the network by operating at a throughput vector x. We will exploit the premise of model assumption (M2) to restrict the utility to the connected component that includes all legitimate nodes. However, this connected component may also contain some adversarial nodes that choose to conform to the protocol, and may change in topology should an adversarial node change strategies. As in part one, we will define the payoff as the time-average utility accrued by all connected components of legitimate nodes over the execution of the protocol.

Let us denote by $q_p(t)$ the set of concurrent transmission vectors disabled by the adversarial strategy at time $t \in [T_l, T_l + T)$. Let $\mathcal{F}_t := \mathcal{F} \setminus q_p(t)$ denote the set of feasible concurrent transmission vectors that have not been disabled by the adversarial nodes. At any time instant t, let $C_t(p, q_p)$ denote the connected component formed by positive rate edges in $\mathcal{F}_t$ that include the legitimate nodes. There are at most a finite number of such connected components over the execution, that we index from $1, \ldots, M$. Let $C^{(j)}(p, q_p)$ denote the jth such component, and α_j the fraction of the operating lifetime $[T_l, T_l + T)$ over which it was active. Let $x_{C^{(j)}(p,q_p)}$ denote the effective throughput attained by this component. We define the payoff as follows:

$$J(p, q_p) := \sum_{j=1}^{M} \alpha_j U\left(x_{C^{(j)}(p,q_p)}\right). \qquad (1)$$

For brevity, we will not discuss (1) further, but refer the reader to part one for a detailed exposition.

IV. The Protocol

We now describe the protocol at the heart of this paper. The protocol itself, shown in Algorithm 1, is composed of two iterative processes: the first enables an existing subnetwork of nodes to operate reliably at a throughput that yields the optimal payoff within a bounded time-scale; the second, working on an unbounded time-scale enables a group of adjacent subnetworks to detect each other and merge into a single supernetwork. The second process works behind the scenes of the first process, periodically activating itself and taking over operations before returning to the background.

The first process is essentially the protocol described in [3], but the second process includes some new innovations; among them, the notion of a "sentinel", first mentioned in the introduction.We will briefly summarize the first process because the main idea also applies to the second process as well.

A. The Process of Converging to a Utility-Optimal Throughput

The first process is composed of five phases: the initial neighbor discovery phase, the initial network discovery phase, the scheduling phase, the data transfer phase, and the verification phase.

Algorithm 1 The Protocol

```
INITIAL NEIGHBOR DISCOVERY PHASE
INITIAL NETWORK DISCOVERY PHASE
while Count $\leq n_{iter}$ do
    if Count $\mod n_b = 0$ then
        SENTINEL PHASE
    end if
    RECURRENT NEIGHBOR DISCOVERY PHASE
    RECURRENT NETWORK DISCOVERY PHASE
    if Successful Merge then
        Reset clocks and change encryption;
        Count $= 0$;
    else if Adjacent Subnetwork Detected then
        Attempt merge if (MR1) and (MR2) are true
    else if Failed Merge then
        Increment corresponding counter;
    else if Time Expired then
        COMA PHASE
    end if
    SCHEDULING PHASE
    DATA TRANSFER PHASE
    VERIFICATION PHASE
    Count $\leftarrow$ Count $+ 1$
end while
```

1) The Initial Neighbor Discovery Phase: At time $t = 0$ (as measured by its local clock) a legitimate node, which we will call node n_A, turns on and enters the initial neighbor discovery phase. During this phase, node n_A through a handshake mechanism, establishes mutually authenticated link certificates with any neighboring nodes. For now we assume that these neighbors and node n_A are born simultaneously and their local clocks have the same skew; node n_A is synchronized with its neighbors.

2) The Initial Network Discovery Phase: Next, node n_A enters the initial network discovery phase. During this phase, the connected component of legitimate nodes containing node n_A, executes a consensus algorithm to obtain a common view of the lists of neighbors acquired during the preceding phase. As a result, node n_A and the legitimate nodes in its connected component also arrive at a common topological view. The problem of achieving consensus in the presence of adversarial agents is well studied [4]. We will not review the details of the algorithm except to state that consensus is guaranteed in synchronized, connected networks with perfect encryption.

By the end of the network discovery phase, this subset of legitimate nodes born at the same instant as node n_A, now forms a rudimentary network in the sense that the legitimate nodes share a common topological view and a common reference clock. This rudimentary network then enters the scheduling phase, where the real effort to arrive at a utility-optimal throughput begins. If no other nodes were born at the same time as node n_A, then node n_A forms a subnetwork of one node. The second process, which we have yet to describe, enables the merge of two adjacent subnetworks. So without loss of generality, we assume that node n_A is part of a subnetwork composed of at least one node.

3) The Scheduling Phase: The rudimentary subnetwork containing node n_A, now in the scheduling phase, determines a utility-optimal schedule of concurrent transmission vectors based on the capacity region of throughput vectors given by the convex hull of $\{r(c), c \in \mathcal{C}\}$. However, the concurrent transmission vectors in this designated schedule may not be feasible; the legitimate nodes do not know $\mathcal{F}$ a priori.

4) The Data Transfer Phase: Despite these uncertainties, the subnetwork enters the data transfer phase and proceeds to execute the schedule in which designated concurrent transmission vectors are allocated specific intervals of time. Those concurrent transmission vectors that are either non-feasible or disabled by adversarial nodes, will fail to deliver their scheduled messages. The intended destinations record the indices of all scheduled packets that failed to arrive.

5) The Verification Phase: Upon completion of the data transfer phase, the subnetwork enters the verification phase. Each destination node releases the list of packets that failed to arrive to the rest of the network by way of the same consensus algorithm used in the network discovery phase. The legitimate nodes in the network then infer the set of concurrent transmission vectors, denoted by $\mathcal{D}^{(1)}$, to which each of these packets belonged. Using this information, the legitimate nodes construct an updated estimate of the feasible set of concurrent transmissions vectors, denoted by $\mathcal{F}^{(2)}$, where $\mathcal{F}^{(2)} := \mathcal{C} \setminus \mathcal{D}^{(1)}$.

After completion of the verification phase, the second process is activated and steers the subnetwork potentially through a sentinel phase, as well as a recurrent neighbor and network discovery phase. The purpose of these phases, as we shall see, is to allow adjacent subnetworks to merge with the subnetwork containing node n_A. For now, let us assume that no merges occur, and that the composition of the subnetwork remains unchanged. The first process then takes over, bringing the subnetwork back into the scheduling phase.

6) Steady State: The sequence of events that we just described repeats itself, in that the subnetwork updates its estimate of feasible concurrent transmission vectors $\mathcal{F}^{(k+1)}$ after the kth iteration, and $\mathcal{F}^{(k+1)} := \mathcal{F}^{(k)} \setminus \mathcal{D}^{(k)}$. Assuming no merges occur, the subnetwork chooses a utility-optimal schedule in the $k+1$th iteration, based on the capacity region of throughput vectors that lie in the convex hull of $\{r(c), c \in \mathcal{F}^{(k)}\}$. The punchline of the entire process is that only a finite number of iterations are needed before the estimates $\mathcal{F}^{(k+1)}$ correctly describe the set of feasible, non-disabled concurrent transmission vectors. This argument follows from (M1); there are a finite number of modulation schemes and a finite number of possible concurrent transmission vectors. It follows, from arguments laid out in [3], that the subnetwork operates to within an ϵ of the locally optimal throughput in the max-min sense (local optimality implies the optimization excludes the legitimate nodes that don't belong to the subnetwork).

B. The Merge Process

We now address the second process of the protocol, in which adjacent subnetworks independently converging to their own locally optimal throughputs, are able to interrupt their operations and merge into one subnetwork.

There are two obstacles to consider. First, each subnetwork operates independently according to a predetermined schedule and estimate of a common reference clock. The subnetworks are composed of distributed nodes and do not have a centralized controller; when a legitimate sentinel node from subnetwork S_A detects a node from another subnetwork S_B, the other legitimate nodes in S_A remain unaware of S_B until receiving a physically transmitted message from the sentinel or an intermediary through the communication medium. Since the nodes in S_A are already following a pre-determined schedule, the sentinel must wait for an alloted time interval in which it can even begin to alert its comrades about S_B. As a result, there is a delay between the time at which a "sentinel" node detects an adjacent subnetwork and the rest of the nodes are alerted to the detection. Furthermore, the detected subnetwork also operates according to a predetermined schedule of its own. If subnetwork S_A, the initiator, wishes to merge with subnetwork S_B, the target, then S_A must wait until S_B enters a pre-scheduled interval in which merges are expected and allowed. Hence, there is also a lag between the time at which the initiator decides to merge with the target, and the time at which the merge is actually attempted.

The scenario we wish to avoid is a "merge chain", in which S_A attempts to merge with S_B while S_B attempts to merge with S_C. Each merge requires the initiator, in this case S_A, to both adopt the schedule of the target S_B, and wait until the time interval allocated by the target's schedule in which merges can occur. However, S_B may have no idea that it is a target of S_A or that S_A even exists; by virtue of being independent S_B operates according to a different schedule than S_A. Therefore, in the interim S_B could change its schedule to merge with S_C without directly informing S_A and complete the merge before the sentinels in S_A are aware of the change or have time to inform their comrades.

These types of merge failures are particularly undesirable because the victim, in this case S_A, cannot determine, based on the evidence, whether the failure was due to malice or circumstance. In many ways, this situation is reminiscent of the failure of a concurrent transmission vector; post-facto, the network cannot infer whether the vector was disabled or infeasible. The merge process will adopt a familiar solution to the problem by imposing a telescoping window of opportunity in which these failures can occur.

The other obstacle to a merge between two adjacent subnetworks is that the operating lifetime of the first subnetwork, which must be fixed as explained in the introduction, could expire before the nodes in the second subnetwork are born. (We say that the nodes are adjacent in the sense that they share an underlying connected component of legitimate nodes).

We now describe how the second process overcomes these obstacles. The process itself is composed of a recurrent neighbor discovery phase, a recurrent network discovery phase, a sentinel phase, and a coma phase. The recurrent neighbor and network discovery phases correspond to the specially designated intervals in which the subnetwork can "absorb" an adjacent subnetwork attempting to merge. The sentinel phase which, to mitigate the throughput loss, occurs at a lower periodicity than the other phases, is the time interval in which the network ceases transmission and attempts to detect the probe packets emitted by adjacent subnetworks. Finally, the coma phase is the phase of undetermined length in which a network enters after its operating lifetime is close to expiry and there are legitimate nodes still yet to be born.

1) The Sentinel Phase: To explain how the merge process works, we will adopt the viewpoint of some subnetwork S_A. Upon entry, the nodes in S_A adopt the role of a "sentinel" in that they cease transmission completely and listen and record any probe packets emitted during the recurrent neighbor discovery phase of any adjacent network.

A sentinel node that detects probe packets emitted from an adjacent subnetwork, is also able to measure the relative skew of the transmitting node from the time-stamps on the probe packets, by taking the ratio of the difference in receive times with the difference in send times. (We assume the probe packets are time-stamped according to the local reference of the corresponding subnetwork). The sentinel also estimates the relative offset of the transmitting node from these time-stamps, to within $2d_{max}$ where d_{max} is the maximum one-hop delay. The relative skew and offset of the transmitting node allow the sentinel to estimate the reference clock of the adjacent subnetwork, and by extension, predict when the next phases of the adjacent subnetwork will occur.

There are two possibilities to consider. One is that the transmitting node and/or the sentinel itself is adversarial, in which case the estimate of the reference clock of the adjacent subnetwork will be wrong. The second, is that the sentinel node is part of an underlying connected component of legitimate nodes that includes the transmitting node in the adjacent subnetwork. In this case, the estimate of the corresponding reference clock will be correct. We will show that by using a pruning strategy similar to the first process, the sentinels are guaranteed to detect an adjacent subnetwork that shares an underlying connected component of legitimate nodes, and correctly estimate the corresponding reference clock, within a finite number of attempts.

The sentinel phase occurs for a time interval equivalent to the length of a full protocol iteration, so adjacent subnetworks are guaranteed to be detected.

2) The Recurrent Neighbor Discovery Phase: Next, the subnetwork S_A enters the recurrent neighbor and network discovery phases. These phases emulate the initial neighbor and network discovery phases described earlier; each node in S_A advertises the existence of S_A by broadcasting probe packets containing the identity certificates of all the constituent nodes. Simultaneously, the nodes from any adjacent subnetworks attempting to merge with S_A also emit corresponding probe packets of their own. The broadcasts are carried out via orthogonal MAC codes, such as the Gold code for example, so that the nodes form either network which may not be perfectly synchronized, can still communicate despite the half-duplex constraints.

3) The Recurrent Network Discovery Phase: The nodes in S_A that are adjacent to nodes in any such external networks, create mutually authenticated link certificates with their adjacent counterparts, that contain their measured relative offsets. Each node in the collection of adjacent subnetworks including S_A shares its newly updated list of neighbors with the rest of the network using a consensus algorithm. By the end of the recurrent network discovery phase, the underlying connected component of legitimate nodes shared by the collection of adjacent subnetworks, obtains a common view of the topology and the relative clock parameters, and establishes a common virtual reference clock. If on the other hand, no subnetworks attempted to merge with S_A, the composition of S_A remains the same as before.

4) Completing a Merge: At this point in the merge process, the subnetwork, now $\tilde{S}_A$, there are four outcomes that dictate the subnetwork's course of action over the next protocol iteration.

(O1) First, $\tilde{S}_A$, determines whether or not a merge occurred during the previous recurrent neighbor and network discovery phases. This decision is straightforward; "yes", if there are nodes in $\tilde{S}_A$ that weren't in S_A, "no" if otherwise. In the event of a "yes" decision, each node in $\tilde{S}_A$ changes its encryption by switching to a new private encryption key and the network $\tilde{S}_A$ resets the virtual reference clock to zero. This move provides $\tilde{S}_A$ with a full operating lifetime to amortize the throughput loss incurred in the interval preceding the merge, in which $\tilde{S}_A$ was artificially segregated into independent subnetworks. The encryption change is needed to prevent the replay attacks described in the introduction. Since the number of distinct merges is finite in network with a finite number of nodes, only a finite number of encryption changes need occur until all the legitimate nodes are finally included in one "super" network.

(O2) Second, $\tilde{S}_A$ examines the list of adjacent subnetworks discovered by the nodes in S_A during the previous sentinel phase. An adjacent subnetwork, call it S_B, is deemed an eligible target for a merge only if certain conditions, which collectively make up "the merge rule", are satisfied. **(MR1)** The *index* of S_B is lower than the index of S_A. (Prior to the start of operations, the protocol assigns an index or order to each of the 2^n possible subnetworks that could be possibly form out of a collection of n nodes, where the index of a subnetwork is greater than or equal to the number of nodes it contains.) Let k denote the index of S_B and let $e := (i, j)$ denote the edge that detected S_B; node $i \in S_A$ received the probe packet broadcast by node $j \in S_B$. **(MR2)** The total number of previous merge attempts between S_A and S_B instigated by edge e does not exceed $k!$. We will show in the proof that (MR1) and (MR2) prevent adjacent subnetworks from getting entangled in an endless cycle of spurious merge attempts with adversarial nodes. The first condition prevents two subnetworks from targeting each other, and the possibility of merge "cycles", whereas the second imposes a ceiling on the maximum number of times a merge between adjacent subnetworks can fail for innocuous reasons.

After choosing a target from the eligible list, $\tilde{S}_A$ makes an estimate of the schedule of the target (S_B) based on the time-stamps received by node $j \in S_B$. Finally, $\tilde{S}_A$ changes its schedule to that of the estimate to ensure that $\tilde{S}_A$ and the target enter the recurrent neighbor discovery phase simultaneously.

The next decision is pertinent if the pre-merge network S_A was the initiator of a merge that failed to absorb any of the nodes in the targeted network.

(O3) If $\tilde{S}_A = S_A$ and S_A initiated a merge attempt during the previous neighbor discovery phase (an attempt that must have clearly failed), then $\tilde{S}_A$ increments the counter of failed merges corresponding to (S_A, e, S_B) by one.

We will show that a utility-optimal super network must eventually form, so long as each transitory legitimate subnetwork keeps a proper account of failed merges.

The last outcome concerns the possibility that the operating lifetime of $\tilde{S}_A$ expires before some legitimate nodes are born, leaving $\tilde{S}_A$ with no avenue through which the virtual reference clock can be reset (a clock reset is always accompanied with an encryption change following a successful merge between two adjacent subnetworks).

(O4) If the virtual reference clock of $\tilde{S}_A$ is about to expire, then each constituent node disassociates itself from $\tilde{S}_A$ (in the sense that the nodes are no longer obliged by the protocol to participate further in $\tilde{S}_A$) and enters the coma phase.

5) The Coma Phase: Upon entry into the coma phase, each node freezes its clock and enters a prolonged period of sentinel duty. A node, call it n_A, remains in the coma phase until it picks up probe packets from an adjacent network containing previously undetected nodes. After detecting an adjacent network, S_B, node n_A unfreezes (not resets) its clock and attempts to merge with S_B as per the procedure described in (O2) and (O3). If the merge fails, n_A returns to the coma phase and refreezes its clock. The left over time in n_A's local clock prior to its first entry into the coma phase be large enough to accommodate the maximum number of possible failed merges that could subsequently occur.

C. The Overall Operation

Having described the protocol operation over the timeline of a single iteration, we now provide a moving snapshot of the topology of the network over an unbounded timeline, from the birth of the first legitimate node until the expiry of the last.

The network begins as a distributed collection of legitimate and adversarial nodes, in which the legitimate nodes are powered off. After the primordial birth of a subset of the legitimate nodes, the first rudimentary subnetworks begin to form composed of legitimate and adversarial nodes. Although the complete set of legitimate nodes, when active, form a connected component, the legitimate nodes within a subnetwork may not be connected. Each subnetwork begins to converge independently on a throughput that is locally optimal with respect to the max-min payoff. Meanwhile, the adversarial nodes dispersed throughout the subnetworks begin to actively disable concurrent transmission vectors (the protocol effectively prevents concurrent transmission vectors from being passively disabled). As a result, some of the subnetworks may fragment into even smaller subnetworks containing connected subcomponents of legitimate nodes.

The individual subnetworks continue to operate independently until the birth of some legitimate node that connects an otherwise disconnected pair. As more and more legitimate nodes enter into the network, disconnected subnetworks become adjacent subnetworks, then detect each other, and begin to attempt to merge. The subnetworks that share an underlying connected component of legitimate nodes eventually merge, possibly after numerous attempts. After each successful merge, the newly consolidated subnetwork starts to converge to its new max-min utility optimal throughput.

The adversarial nodes in the meantime, attempt to entice the subnetworks into making spurious merge attempts, by broadcasting false or misleading probe packets. However, every failed merge attempt reduces the credibility of the node pair that instigated the attempt. The adversarial nodes are forced into a telescoping window of opportunity to disrupt the merging of adjacent subnetworks. As time progresses, subnetworks emerge, fragment, reconstitute, merge, freeze altogether, then restart, but eventually form a "super" network containing all legitimate nodes. This super network operates at a min-max optimal throughput, evaluated from the birth of the last legitimate node.

V. Main Results

Let (p_1^*, q_1^*) be the solution to the following optimization problem:

$$\max_{\text{protocols } p} \min_{\text{attacks } q_p} J(p, q_p). \tag{2}$$

Let $x_1^* := f(p^*, q_p^*)$, where f denotes the physical properties of the channel as specified in the model assumptions, and x is the throughput vector evaluated over the interval $[T_l, T_l + T)$. The precise relationship between x and (p, q_p) is described in part one. We have the following result:

Theorem 1. *Given any $\epsilon > 0$, the protocol described achieves a throughput $(1-\epsilon)x_1^*$, evaluated from the birth of the last legitimate node.*

In fact, we have a stronger result. Suppose we redefine the game so that the adversarial nodes first choose a strategy q and the protocol p is chosen with a priori knowledge of q. Let $\tilde{\mathcal{D}}$ denote the set of strategies q in which the adversarial nodes are limited exclusively to actively disabling concurrent transmission vectors at each time instant t. Let (p_2^*, q_2^*) denote the solution to the following optimization problem:

$$\min_{\text{attacks } q} \max_{\text{protocols } p} J(p, q). \tag{3}$$

Let $x_2^* = f(p_2^*, q_2^*)$. We have the following result:

Theorem 2. *Given any $\epsilon > 0$, the protocol achieves a throughput $(1-\epsilon)x_2^*$, evaluated from the birth of the last legitimate node.*

The significance of Theorem 2 is that the adversarial nodes gain no advantage from knowing the protocol a priori. Moreover, the strategies of the adversarial nodes are effectively limited to actively disabling concurrent transmission vectors, which in effect, corresponds to either jamming or conforming with the protocol. Both theorems were proved in [3] for closed networks.

VI. Proof of Theorem 2

The proof of Theorem 2 is divided into four parts for ease of presentation. First, we argue that given model assumptions (M1)-(M4), the protocol described in Section IV enables a distributed collection of nodes to form a rudimentary subnetwork in which the legitimate nodes share a common topological view and a virtual reference clock. Second, we argue that this rudimentary network, barring merge induced interruptions, will operate at a min-max optimal throughput x_2^* with respect to the payoff $J(p, q_p)$, evaluated from the birth time of the last legitimate node in the subnetwork, regardless of what the adversarial nodes do. Both claims are similar to proofs in [3] so we do not relieve them here.

Next, we show that the protocol, using a bounded number of merge attempts, enables a collection of adjacent subnetworks to merge into one subnetwork containing the underlying subcomponent of legitimate nodes. Finally, we show that the throughput loss incurred from the protocol overhead, failed merge attempts, and sub-optimal pre-merge operation, is an arbitrarily small fraction of the operating lifetime, evaluated from the birth time of the last legitimate node.

Let S_i denote the subnetwork of index i. Let $S_{i_1}, \ldots, S_{i_m}$, where $i_1 < i_2 < \ldots < i_m$, be the collection of all subnetworks containing legitimate nodes that formed during the execution of the protocol p and adversarial response q_p during $[0, T_l + T)$ (the period of time, with respect to the global reference clock, from the birth of the first legitimate node until the expiry of the last). Let $M(p, q_p)$ be the total number of merge attempts that occurred during $[0, T_l + T)$.

Lemma 1. *The subnetwork S_{i_m} contains the underlying connected subcomponent of legitimate nodes, and $M(p, q_p) \leq (i_m!)^3$.*

Proof. Let S_i and S_j denote a pair of adjacent subnetworks where $i > j$. Let e be a connecting edge between S_i and S_j where e may or may not have adversarial endpoints. We will show, (a) that for each e, the subnetwork S_i needs to make at most $(j!)^3$ merge attempts with S_j. The proof of (a) is by induction. If $j = 1$ then S_j does not attempt any merges since there are no subnetworks of lower index with which it can merge. Now suppose e has legitimate endpoints; either S_i successfully merges with S_1 on the first attempt or S_1 merges with another subnetwork and ceases to exist as S_1. In either case, S_i need only attempt one merge with S_1 per edge e. Now we assume the statement (a) is true for $j - 1$ and show that (a) also holds for j. For $j > 1$, there are subnetworks of lower index than j with which S_j could attempt to merge. By the induction hypothesis, S_j needs to make at most $(k!)^3$ merge attempts for each connecting edge with S_k where $j > k$. There are at most j^2 edges connecting S_j with any such S_k, since by construction, the index of a subnetwork is greater than the number of nodes in the subnetwork. The total number of merge attempts that S_j will make is $\sum_{k=1}^{k=j-1} j^2 (k!)^3 < j^2 \sum_{k=1}^{k=j-1} (k!)^3 < j^2 (j-1)((j-1)!)^3 < (j!)^3$. It follows that S_i need only attempt $(j!)^3$ merge attempts with S_j per edge e. Therefore (a) is proved and the lemma follows as a consequence. □

Finally we show that the total throughput loss, evaluated from the moment of the birth of the last legitimate node can be made arbitrarily small.

Lemma 2. *The throughput achieved by the protocol is $(1 - \epsilon_l)(1 - \epsilon_m)(1 - \epsilon_s)x_2^*$, where ϵ_l, ϵ_m, and ϵ_s can be made arbitrarily small.*

Proof. The first part of the lemma, the claim that the subnetwork S_{i_m} has an effective throughput of $(1 - \epsilon_l)(1 - \epsilon_m)(1 - \epsilon_s)x_2^*$, essentially follows from the arguments in [3]. We will show that ϵ_l, ϵ_m, and ϵ_s can be made arbitrarily small. The "long-run" throughput loss, ϵ_l is the loss incurred by sub-optimal operation prior to a merge between two adjacent subnetworks, as well as the use of disabled concurrent transmission vectors. Let n_b denote the number of blocks of protocol iterations, where a "block" denotes the number of protocol iterations that occur between sentinel phases. By Lemma 1 the total number of merges that occur is at most $(i_m!)^3$, where $i_m < 2^n$. By model assumption (M1) there are at most N concurrent transmission vectors. Therefore, choose n_b so that $\epsilon_l = \frac{\max\{N, ((2^n)!)^3\}}{n_b}$. The "medium-term" throughput loss ϵ_m is the loss incurred by periodically detouring into the sentinel phase for a protocol iteration. Let n_s be the number of protocol iterations between each sentinel phase. Choose n_s so that $\epsilon_m = \frac{1}{n_s}$. (The total number of protocol iterations n_{iter} in Algorithm 1 is $n_{iter} := n_s n_b$.) The "short-run" throughput loss, denoted by ϵ_s, is the overhead loss incurred during a protocol iteration from the phases in which data was not transfered. We show in [3] that the short-run throughput loss can be made arbitrarily small. □

VII. Conclusion

We proposed a protocol with a provable guarantee of security and performance for open, unsynchronized networks. We showed that the key idea in part one also applies here: the adversarial strategies are confined to a finite set that is successively whittled away after each protocol iteration. In future work, we would like the model to include probabilistic packet loss and channel fading.

References

[1] D. Kuipers and M. Fabro, *Control systems cyber security: Defense in depth strategies.* United States. Department of Energy, 2006.

[2] Y.-C. Hu, A. Perrig, and D. B. Johnson, "Ariadne: A secure on-demand routing protocol for ad hoc networks," *Wirel. Netw.*, vol. 11, no. 1-2, pp. 21–38, Jan. 2005.

[3] J. Ponniah, Y. Hu, and P. R. Kumar, "A system-theoretic clean slate approach to provably secure ad hoc wireless networking," *IEEE Transactions on Control of Network Systems, To appear.*

[4] L. Lamport, R. Shostak, and M. Pease, "The byzantine generals problem," *ACM Trans. Program. Lang. Syst.*, vol. 4, no. 3, pp. 382–401, Jul. 1982.