Symphony: Cooperative Packet Recovery over the Wired Backbone in Enterprise WLANs

Tarun Bansal†, Bo Chen†, Prasun Sinha and Kannan Srinivasan
Department of Computer Science and Engineering
The Ohio State University
Columbus, OH 43210
{bansal, chebo, prasun, kannan}@cse.ohio-state.edu
†Co-primary authors

ABSTRACT

In this paper, we propose Symphony, a packet recovery architecture that encourages collisions among transmitters, and utilizes the unused capacity in the backbone to transmit recovered data packets and coordinate the efficient recovery of collided packets. Symphony improves the wireless throughput while incurring a low overhead on the typically under-utilized wired backbone. In Symphony, upon receiving the collided transmissions, the APs carefully suppress a subset of the transmissions. Realizing this idea in practice entails several challenges including identification of clients that have data to transmit and ensuring that the algorithm works despite imperfect time-synchronization and non-zero latency among APs. We present Symphony that addresses these challenges and show how it leverages Successive Interference Cancellation (SIC) to further increase the network throughput. Experiments performed on an USRP testbed shows that on an average, Symphony provides 43% and 187% higher throughput over Omniscient TDMA and IEEE 802.11, respectively. ns-3 based simulation results show that on an average, Symphony provides a throughput of up to 1.63x compared to omniscient TDMA and 5.6x compared to IEEE 802.11.

Categories and Subject Descriptors

C.2.2 [**Computer-Communication Networks**]: Network Protocols

Keywords

Wireless Networks; Cooperative Decoding

1. INTRODUCTION

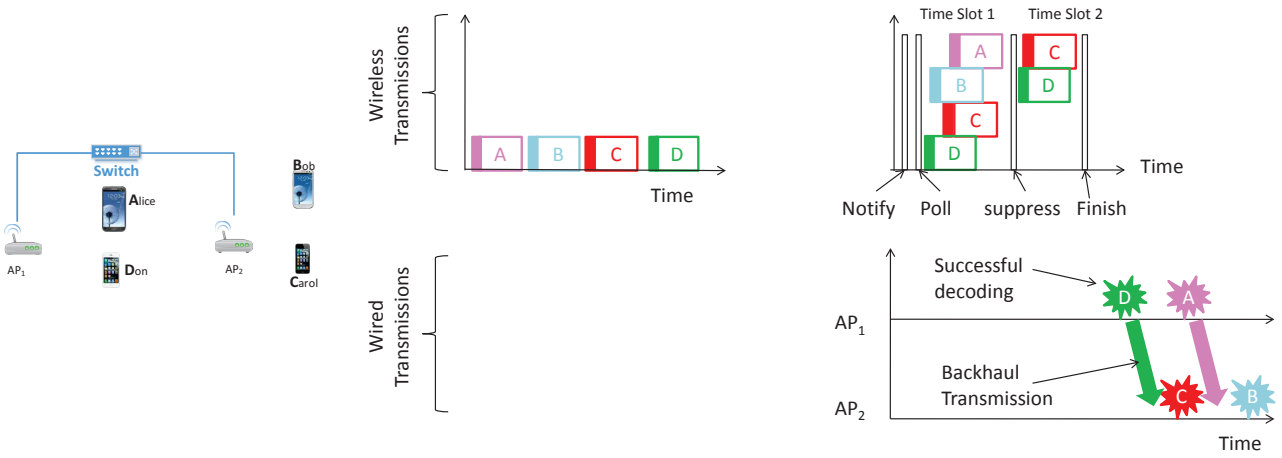
APs in Enterprise Wireless Local Area Network (EWLAN) are typically connected using a wired backbone. *In this paper, we propose Symphony, a cooperative decoding approach that minimally uses the wired connection among APs to increase the throughput of wireless networks.* Symphony encourages collision of packets among transmitters at APs. On receiving multiple packets, APs suppress a subset of colliding transmitters. This reduces the number of transmitters in each slot. Eventually, the number of transmitters reduces to a small enough value such that the APs can cooperatively decode all the received packets.

Figure 1a shows a simple topology with two APs and four clients where the APs are connected to each other through a wired backbone. Assume that each of the clients needs to upload one packet to the backbone. For this topology, omniscient TDMA will take at least four slots to schedule the four transmissions (See Figure 1b). In Symphony, on the other hand, all the four clients transmit (See Figure 1c) in the first slot. At the end of the slot, Symphony suppresses Alice (A) and Bob(B). In the next slot, only Carol(C) and Don(D) transmit. At the end of the second slot, AP_1 decodes the packet D since AP_1 received it interference-free. After decoding, it sends the decoded bits of D to AP_2 over the wired backbone. From the received bits, after correcting for different offsets, AP_2 recreates samples received from D and subtracts them from the samples received in slot 2. After subtraction, it is left with only the samples corresponding to C which AP_2 decodes. At the same time, AP_1 recreates the samples of D as received in slot 1 and subtracts those samples from the samples received in slot 1. AP_1 then decodes the remaining samples to obtain A and sends it to AP_2. Finally, AP_2 recreates the received samples of A, C and D for slot 1. After subtracting these samples from the samples received in slot 1, AP_2 decodes B. Using cooperation among APs and by using the backbone for exchanging decoded packets, Symphony enables the two APs to receive the four packets in two slots. Thus, by utilizing cooperation among APs to simultaneously decode multiple colliding transmissions, Symphony provides twice the throughput as compared to omniscient TDMA.

Cooperation among APs has been used before to increase wireless throughput. Authors in [17] focus on improving the client-AP association and AP channel assignment. Miu *et al.*[16], Woo *et al.* [22]and Gowda *et al.* [10] have respectively proposed techniques for bit-level, symbol-level and coarse symbol-level combining across different APs. In these solutions, a packet overheard by different APs is decoded cooperatively by multiple APs. However, in all these algorithms, all the APs cooperate to decode a single packet while in Symphony, APs cooperate to simultaneously de-

MobiCom'13, September 30–October 4, Miami, FL, USA.

http://dx.doi.org/10.1145/2500423.2500450.



(a) Network Topology (b) Omniscient TDMA (c) Symphony

Figure 1: Network topology and set of packets received by APs. (a) Network topology. AP_1 is assumed to be outside the interference range of Bob and Carol. (b) Omniscient TDMA takes a minimum of 4 slots. (c) Symphony can receive four packets in two slots. In Symphony, APs cooperatively decode packets in the reverse chronological order by first decoding the packets received in the last slot.

code multiple packets. Further, symbol-level combining[16] across different APs is prohibitive due to significantly higher bandwidth requirements[22, 10] on the wired backbone. In Symphony, APs only exchange decoded packets which are typically much smaller in size compared to raw samples [22, 10]. The idea of exchanging packets on the backbone was also studied in [9]. However, unlike Symphony, [9] provides throughput improvement only when all the devices have at least two antennas.

Several prior works [21, 18, 14] have focused on increasing network throughput for downlink traffic in EWLANs while neglecting the uplink traffic. On the other hand, Symphony primarily improves the network throughput on uplink while allowing efficient schemes such as conflict-free TDMA scheduling[14] to be utilized over the downlink. We believe that upstream traffic is increasing rapidly due to rise in use of a wide-range of new storage paradigms, computing paradigms and applications, such as cloud computing, cloud storage, P2P file access, video conferencing, online gaming, VoIP, and traffic generated from the mobile devices (e.g., location information or sensor readings). Using cooperation to improve uplink wireless throughput is particularly challenging, since APs are not aware of which client has data to transmit making it difficult to schedule transmissions. Collecting this information from different clients is also cost-prohibitive.

Although the idea behind Symphony is simple, there are multiple challenges to realize it in practice.

1. **Set of nodes to suppress:** At the end of each slot, Symphony needs to determine the set of nodes to be suppressed. This set should be feasible (ensuring that the set of suppressed transmissions would be decoded in future) and large, so that all the collided transmissions can be decoded in a small number of slots.

2. **Synchronization and variable non-zero latency:** Cooperation among APs requires that APs be synchronized with each other and that their mutual-latency be low and predictable. However, our measurements described in Section 7.2 show that these requirements do not hold true in large networks.

3. **Absence of central controller:** In wireless networks, it may not be possible to set up a central controller that coordinates all the APs in the entire network due to the large size of the network. Thus, it is required that cooperative decoding should work in the absence of central controller.

4. **Knowledge of topology:** To cooperatively suppress a subset of transmitters, the APs need to know the network topology. This information (particularly the interference among non-AP nodes) is difficult to acquire for large scale networks [14, 9] and could lead to significant overhead.

5. **Difference in packet lengths:** It is possible that different transmitters may have different amount of pending data left. However, if the channel access time is fixed as shown in Figure 1c, then the users with less pending data may not fully use the access time leading to channel wastage.

The paper is organized as follows. Section 2 presents a brief overview of Mozart[2], a recently proposed collision recovery based MAC layer protocol. The next section describes a centralized algorithm that assumes the presence of a central server. In Section 4, we describe our distributed protocol, Symphony that handles all the above challenges. In Section 5, we explain how Symphony harnesses SIC to increase wireless throughput. The following section discusses some issues that makes our solution more practical. Section 7 and Section 8 outline the results from our USRP testbed experiments and ns-3 based simulations, respectively. Section 9 presents the state of the art and finally, Section 10 concludes our paper.

2. BACKGROUND

A recently proposed protocol, Mozart [2] uses packet subtraction and receiver controlled retransmissions to decode collided packets in the least possible number of slots. At

the end of each slot, receivers in Mozart suppress one transmitter independently. In order to suppress some transmitter, receivers in Mozart need to identify which set of clients are transmitting. In [2], authors propose that transmitters should use Pseudo-random (PN) sequences to identify themselves. The PN sequences (implemented using Gold codes [7]) can be correlated by the receiver even under high interference [15]. Thus, by correlating received samples, the receivers in Mozart identify the IDs of as many colliding transmitters as possible. To harness SIC, Mozart's receivers need to estimate the individual RSS of each of the colliding transmitters. Mozart proposes three different approaches (PN sequence padding, cancellation, iterative estimation) which when combined significantly improve the accuracy of transmitter as well as the RSS estimation accuracy. In this paper, we use the methods proposed in Mozart [2] for sender identification and RSS estimation. Observe that *unlike Mozart, Symphony uses the wired backbone to cooperatively decode the collided packets.* This leads to a different set of challenges as described in Section 1.

3. COOPERATIVE PACKET SUBTRACTION

In this section, we present a simplified version of our algorithm, called Centralized Algorithm (CA) that works under a centralized framework. In Subsection 3.2, we explain the practical limitations of CA.

CA assumes the presence of a central server (CS) that coordinates different APs. CA also assumes that all the APs are synchronized with the CS and their mutual latency is zero. We relax these assumptions in Section 4 where we describe our distributed algorithm. CA works in multiple phases as described below (See Figure 1c):

1. **Polling:** First, using the wired backbone, the CS asks all the APs to simultaneously transmit poll packets on the air. The poll packets are encoded using PN sequences [15]. Use of PN sequences, allows the clients to correlate them even if multiple poll packets collide.

2. **Data Transmission:** Upon receiving the poll, the clients that have uplink data transmit their data packets. The clients also encode their IDs using their PN sequence and include it in the packet's header at the front. Due to simultaneous transmissions, multiple packets may collide at the APs. As explained in Section 2, each AP uses the PN sequence to determine the ID of as many colliding transmitters as possible, and then forwards this information to the CS using the wired backbone.

3. **Computing the set of suppressed clients:** On receiving the *best-effort* information about which AP received packets from which clients, the CS uses Algorithm 1 (explained in Section 3.1) to compute the subset of transmitters to be suppressed at the end of this slot. The algorithm described in Section 3.1 also determines which AP should suppress which client. All this information is conveyed by CS to the APs through the backbone.

4. **Transmitting the suppress packets:** Upon receiving the information about which nodes to suppress, the APs transmit the *suppress* packet that includes the PN sequence of the client that is suppressed. Similar to poll packets, here also use of PN sequences allows the clients to correlate them even if multiple suppress packets collide.

5. **Data retransmission:** Each client upon hearing the suppress determines if it is being suppressed: If so, it does not transmit (until it receives *finish*), otherwise, the client retransmits its data.

6. **Finish:** As the nodes are suppressed, the number of concurrent transmissions received by APs decreases with each slot. Eventually, it becomes possible for the APs to use cooperation and decode all the transmissions received in the last slot. As described in Section 1, this decoding is done in the *reverse chronological order* until all the received packets are decoded. At that point, the CS directs all APs to broadcast *finish*. The reception of a finish packet at the clients indicates that the data transmitted by the nodes has been successfully received at APs. This allows the nodes to transmit the next MAC-layer data frame.

Another way of scheduling transmissions may be that in the first slot, using the poll, APs gather the PN sequences of all the clients that have uplink data and then schedule different transmitters in a TDMA fashion. However, this approach does not work well since APs may not detect PN sequences from transmitters that have low SINR[2]. Thus, transmitters with low SINR will not be scheduled and get starved. On the other hand, the PN sequences of such transmitters will be detected in CA in later slots when the number of colliding packets become small. So, with the reverse chronological decoding of Symphony, the transmitters with low SINR will also be suppressed and eventually their packets would also be decoded.

To prevent overwhelming[10, 22] the wired backbone, one of the requirements for cooperative decoding is that APs should not exchange sample level information on the wired backbone[1]. The set of transmissions suppressed during a slot and the samples received at all the APs during that slot form a set of linear equations. Observe that, just because the set of equations is solvable, it does not imply that all the suppressed transmissions can be decoded without exchanging sample level information. Consider the example network shown in Figure 1a. If at the end of slot 1, the CS suppresses A and D, then even though the set of linear equations has 2 variables (A and D) and 2 equations (samples received at AP_1 and AP_2), still it is not possible to decode both A and D *without the exchange of samples.* Thus, we need another mechanism to compute the set of suppressed transmissions.

3.1 Computing the set of suppressed transmissions

At the end of each slot, the CS computes the set of suppressed transmissions. *This set should satisfy the following four requirements*: (*i*) **Feasibility:** It should be a feasible set such that the APs can successfully decode all the suppressed packets without exchanging sample-level information; (*ii*) **Optimality:** It should maximize the number of transmissions suppressed in this slot; (*iii*) **Optimality in future slots:** It should maximize the applicability of cooperative decoding in future slots; and, (*iv*) **Low backbone overhead:** It should minimize the number of decoded packets that need to be exchanged among APs, thereby reducing the overhead on the wired network.

[1]Although by exchanging coarse samples [10], it is possible to reduce the overhead in the backbone, it is not enough for our purpose since high fidelity representation of symbols is required to decode a packet in the presence of interference from other transmissions.

Algorithm 1: Computes the set of nodes to be suppressed in a given slot

1 **Input:** For every packet P_i received in this slot by each AP AP_j, the transmitter of P_i (identified using PN sequence).
2 **Output:** The acyclic subgraph showing: Set of packets (or corresponding transmitters) that should be sent suppress in this slot, which AP decodes which packet and how APs share decoded packets among themselves.

`// Phase 1: Computing dependence graph` $G_d = (V_d, E_d)$
3 $V_d \leftarrow \{v_{ij} : P_i$ received in this slot and AP_j can decode P_i in absence of external interference$\}$
4 $E_d \leftarrow \{(v_{ij}, v_{kl}) : v_{ij} \in V_d$ and $v_{kl} \in V_d$ and P_i interferes at AP_l and $v_{ij} \neq v_{kl}\}$

`// Phase 2: Computing largest IDAS`
5 $V_s \leftarrow \phi, E_s \leftarrow \phi$, $G_s \leftarrow (V_s, E_s)$
6 **while** *true* **do**
7 $\quad v^* \leftarrow \phi, w^* \leftarrow \infty, E^* \leftarrow \phi$
8 $\quad$ **for** $v_{ij} \in V_d \backslash V_s$ **do**
9 $\quad\quad E_{ij} \leftarrow \{(v_{ij}, v_{kl}) : (v_{ij}, v_{kl}) \in E_d \wedge v_{kl} \in V_s\}$
10 $\quad\quad E_{ij} \leftarrow E_{ij} \cup \{(v_{kl}, v_{ij}) : (v_{kl}, v_{ij}) \in E_d \wedge v_{kl} \in V_s\}$
11 $\quad\quad$ **if** $w_{ij} \leq w^*$ *and graph with* $V_s \cup \{v_{ij}\}$ *and* $E_s \cup E_{ij}$ *is acyclic* **then**
12 $\quad\quad\quad w^* \leftarrow w_{ij}, v^* \leftarrow v_{ij}, E^* \leftarrow E_{ij}$
13 $\quad$ **if** $v^* \neq \phi$ **then**
14 $\quad\quad V_s \leftarrow V_s \cup \{v^*\}$`// Add` v^* `to` G_s
15 $\quad\quad E_s \leftarrow E_s \cup E^*$
16 $\quad$ **else**
17 $\quad\quad$ **break**
18 **return** G_s

Computing the largest set of feasible suppressed transmissions (while taking into account only the first two requirements) is a NP-Hard problem (shown in the technical report [1]). To that end, we propose a two-phased greedy algorithm that computes the set of suppressed transmissions by first constructing a *dependence graph* (Explained in Subsection 3.1.1) and then picking its maximum acyclic induced subgraph (Explained in Subsection 3.1.2). This acyclic graph corresponds to the set of transmissions that can be successfully suppressed in this slot. In this paper, we use P_i to denote packet received from client i. We overload this term and also use P_i to represent the client that transmitted the packet.

3.1.1 Phase 1: Constructing the dependence graph

When using cooperative decoding, decoding of one transmission may depend on successful decoding of another transmission. For the example network shown in Figure 1a, let's say that in the first slot, the CS suppresses A and B. However, since A's transmission interferes at AP_2, therefore, B's transmission at AP_2 can be decoded only after the former transmission has been decoded at AP_1. In general, for two transmissions $\{P_i, AP_j\}$ and $\{P_k, AP_l\}$, if P_k interferes at AP_j, then the transmission from P_i can be decoded at AP_j only after P_k's transmission has been decoded. Thus, we say that the decoding of P_i's transmission is *dependent* upon the decoding of P_k's transmission. In this phase, the CS constructs a *dependence graph* for the current slot (Lines 3-4 of Algorithm 1)):

1. **Vertices:** Corresponding to each client-AP link in the topology, a vertex is created in the dependence graph. We use v_{ij} to denote the vertex that corresponds to the link $\{P_i, AP_j\}$. v_{ij} is created *iff* P_i transmitted at least one packet in this slot and AP_j is in the receiving range of P_i (i.e., AP_j can decode P_i in the absence of interference).

2. **Edges:** Edges in the dependence graph capture the decoding dependence among different transmissions. A directed edge is added from vertex v_{ij} to a vertex v_{kl} *iff* (i) P_i interferes at AP_l; and, (ii) v_{ij} is not same as v_{kl}. The second condition forbids self-loops in the graph.

By creating vertices on the basis of transmission range and edges on the basis of the interference range, the algorithm is able to handle the differences between the transmission and interference range of practical radios. Figure 2a shows the dependence graph for the example network shown in Figure 1a at the end of slot 1. In the following discussion, we use G_d to denote the dependence graph.

3.1.2 Phase 2: Computing the maximum vertex-induced directed acyclic subgraph (IDAS)

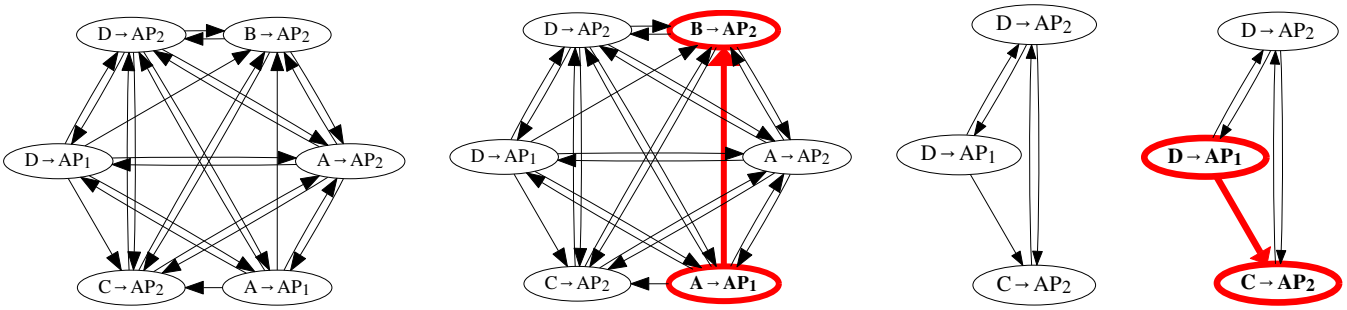
We prove (See technical report [1]) that a set of client-AP transmissions can be decoded if and only if the vertex induced subgraph of the corresponding set of vertices in G_d is a DAG (Directed Acyclic Graph). For example, since transmissions $A \rightarrow AP_1$ and $B \rightarrow AP_2$ (See Figure 2a) are part of induced acyclic subgraph of the dependence graph, therefore both these transmissions can be suppressed at the end of slot 1. Thus, to compute the largest set of suppressed nodes in this slot, the CS computes the maximum Induced Directed Acyclic Subgraph (IDAS) of G_d. Algorithms for computing the maximum IDAS have been proposed before [6]. Although, those algorithms take into account the first and the second requirements discussed in Section 3.1, but they do not take into account the other two requirements. To that end, we now explain the greedy algorithm used by CS to compute the IDAS (denoted by $G_s = (V_s, E_s)$) of G_d.

CS works by greedily adding vertices while ensuring that G_s stays acyclic. At any step, it may be possible to add any of the multiple vertices (each vertex corresponds to some client-AP pair in the dependence graph) available to G_s. However, different vertices may give different performance results. To that end, for each vertex that can be added to V_s, CS computes its *weight* and greedily adds that vertex that does not create a cycle in G_s and has the lowest weight (Lines 8-15 of Algorithm 1). This process is repeated until no more vertices can be added to G_s (Lines 6-17 of Algorithm 1).

Computing the weight: When adding a vertex (say v_{ij}) to G_s, the following equation is used to compute its weight:

$$\begin{aligned} & w_{ij} = w(P_i, AP_j) \\ = \; & |(v_{ij}, v_{kl}) : (v_{ij}, v_{kl}) \in E_d \wedge v_{kl} \in V_s| \\ & \times |(v_{kl}, v_{ij}) : (v_{kl}, v_{ij}) \in E_d \wedge v_{kl} \in V_s| \end{aligned} \quad (1)$$

If on adding v_{ij} to V_s, it has either fewer incoming edges or fewer outgoing edges, then this indicates that v_{ij} has low probability of being part of some cycle in the future. This increases the possibility of extending G_s to a larger acyclic induced subgraph in the following iterations. Further, fewer edges indicate that APs need to exchange fewer packets in the backbone. Thus, (1) assigns a lower weight to the vertices with either fewer incoming edges or fewer outgoing edges and thus, a higher priority. Therefore, by using (1), Algorithm 1 tries to take all the requirements into account that were discussed in Section 3.1.



(a) G_d at the end of slot 1

(b) Two vertices and one induced edge (shown in bold) form the Acyclic subgraph (G_s) at the end of slot 1.

(c) G_d at the end of slot 2

(d) G_s (shown in bold) at the end of slot 2.

Figure 2: Dependence graph (G_d) and Induced Directed Acyclic Subgraph (IDAS denoted by G_s) at the end of the first and the second slot for the example network shown in Figure 1a.

Figure 2b shows the computed IDAS for the dependence graph shown in Figure 2a. The subgraph shows that AP_1 suppresses A while AP_2 suppresses B. After the APs send the suppress, only C and D transmit in the second slot. Figure 2c shows the dependence graph created by CA at the end of the second slot. Figure 2d shows the computed IDAS which shows that C and D are suppressed in this slot. Since, no other transmitters are left for transmitting, so, the CS would ask APs to send *finish*. Observe that the edges in the G_s also reveal how the APs should exchange the decoded packets. For example, the directed edge from "$D \rightarrow AP_1$" to "$C \rightarrow AP_2$" indicates that decoding of C at AP_2 is dependent upon decoding of D at AP_1. Thus, AP_1 should send the decoded bits of D to AP_2. The CS transmits this information to the APs so that they only exchange packets that are required, thereby reducing the overhead on the backbone.

3.2 Practical Challenges

Although the algorithm proposed before is good in theory, it is difficult to implement due to various challenges:

1. **Synchronization:** CA requires that all APs be time-synchronized to ensure correct correlation of PN sequences. As described in Section 2, receivers in CA use correlation of Gold codes to identify the set of transmitters. Gold codes guarantee that the circular cross-correlation of two instances of Gold code is bounded. For 127-symbol sequence Gold codes, the bound is 1/7[7]. So, the correlation works better when Gold codes collide only with other Gold codes, and not with arbitrary samples from data packets. This implies that CA needs to ensure that when an AP receives a set of collided transmissions, then the PN sequences in them only collide with other PN sequences. One way to satisfy this requirement is to ensure that all APs in the network poll and suppress in a synchronous fashion. Algorithms for synchronization in wired networks have been proposed before [11, 12], however, as the size of the network grows, ensuring synchronization across the entire network becomes difficult. Thus, a new method is needed to ensure that PN sequences collide only with other PN sequences so that they can be correlated with high accuracy.

2. **Unpredictable delays over the backbone:** In CA, the central server computes the set of suppressed transmitters and distributes that result to the APs. However, due to unpredictable delay, it is possible that this result may not arrive in time and thus, some of the APs may not be able to proceed.

3. **Non-uniform distribution of clients across APs:** The previous algorithm requires that all APs poll at the same time. In an enterprise network, it is possible that different APs have unequal number of neighboring clients with outstanding data. This may result in different APs finishing at different times, leading to lower throughput as all the finished APs in the network need to wait for the AP with the highest number of clients.

4. **Absence of central controller:** As discussed in Section 1, a central controller may not be present to control all the APs in the network.

4. SYMPHONY: DISTRIBUTED ALGORITHM

In this section, we explain the working of Symphony that takes into account the aforementioned challenges in Section 3.2.

4.1 System Structure

Using cooperative decoding requires multiple APs to collaborate, however, at the same time, it may not be possible for all the APs in the network to collaborate since the wired backbone delays from APs to the central controller may be unbounded. Further, if the central controller is overseeing a lot of APs, it may also become a bottleneck. Symphony handles these contradicting requirements by dynamically placing different APs in different *groups*. One AP in each group serves as the Group Head (GH) and executes Symphony independently from other groups. Putting APs in different groups allows Symphony to achieve multiple objectives: (i) When dynamically creating groups, it is ensured that the average round trip delay on the wired backbone from the AP to the GH is no more than a threshold (ψ μs), thereby bounding the expected communication delay between the AP and the GH with high probability; (ii) If one AP has high number of clients, then only the APs in the same group are required to wait for this AP to finish; and, (iii) No central controller is required.

Implementing this dynamic group based structure leads to various **challenges**: (i) To avoid decoding failures due to conflicting *suppress* schedules, a client that is in the range

of APs from different groups is not allowed to transmit (discussed below). Thus, it is important to dynamically change the composition of the groups so that all clients get an opportunity to transmit; and, (ii) The overhead of maintaining the groups should be minimized. The next subsection describes the group management algorithm used in Symphony.

4.2 Group Management

Group creation and joining an existing group: When a node (AP or client) turns *on* or whenever the node is not part of any group, it enters the *wait* stage where it waits to hear a *join* message on the wireless. This message includes the PN sequence of the GH that originated this message. In Symphony, the ID of a group is same as the ID of its GH. Therefore, the PN sequence included in *join* is also useful in determining which group a received message came from. If the node (say n_i) hears such a message, it joins the announced group if all of the following requirements are satisfied: (*i*) n_i is not in *recovery*; (*ii*) n_i has no wireless neighbor n_j such that n_j is a member of some other group and is under recovery; and, (*iii*) If n_i is an AP, then the average round trip delay between n_i and the GH on the wired backbone is no more than a threshold ($\psi\mu$s). If n_i determines that it is eligible to join the group, then it immediately rebroadcasts the *join* message. Once n_i rebroadcasts the message, we say that n_i has joined the group and has entered recovery. On the other hand, if n_i is an AP and it does not hear any *join* message within a certain time (θ μs) that satisfies all the requirements, then it exits the wait stage. Next, if n_i checks if it has a wireless neighbor n_j such that n_j is a member of some other group and is under recovery. If n_i has no such neighbor, then n_i creates a new group and declares itself as the GH. Afterward, it broadcasts a *join* message.

Data transmission: A client broadcasts its data packet immediately after broadcasting the *join*. Once the APs receive the headers of the data packets transmitted by clients, the APs correlate the PN sequences in the headers to determine the IDs of clients that transmitted (See Section 2). This information is then sent by the APs to the GH that uses algorithm described in Section 3.1 to compute which AP will suppress which client. This result is then sent back by the GH to all the APs who use this result to suppress the clients at the end of the slot. However, due to unpredictable queuing delays on the backbone, it is possible that some APs do not receive the result back from the GH before the end of the slot. To handle such scenarios, the GH iteratively computes the set of nodes to suppress for the next k slots and sends that to all the APs (We explain in Section 7.2 how k is determined). Further, in the unlikely case when some AP does not hear back from GH for more than k slots, then at the end of the slot that AP does not suppress any client.

Finishing the recovery: As the APs suppress transmitters, eventually the number of packets transmitted will be small enough such that they can be decoded. This marks the end of recovery. At the end of recovery, the APs need to take a few steps to ensure: (i) GH is dynamically changed to ensure that all APs have equal chance of being the GH; (ii) Members belonging to a small group have a non-zero probability of merging with a bigger group; (iii) Members belonging to a large group have a non-zero probability of splitting into multiple smaller groups; and, (iv) Next recovery starts as soon as possible. We now explain the steps taken by APs at the end of the recovery: (*i*) At the beginning of the last slot, the GH sends a message to the APs announcing that the recovery will finish at the end of this slot. This allows the APs to know when to enter the *wait* stage. (*ii*) Before the recovery is over, the GH randomly selects some other AP in the group to be the new GH. The current GH informs this new potential GH of this decision. This step ensures that all APs have equal chance of being the GH. (*iii*) Once the recovery is over, all APs enter the *wait* stage for a duration of θ μs as described in the beginning of Section 4. For all the APs, θ is inversely proportional to the number of APs in their group. This ensures that smaller groups wait for a longer time to receive a *join* message from other groups, and thus they have higher probability of merging into another group. Also, the value of θ is set to be low for the selected GH while it is higher for the other APs. This ensures that the selected GH has higher chances of becoming a GH. This also reduces the probability of multiple APs independently declaring themselves as GH. This allows Symphony to set θ to a low value, thereby reducing the overhead of group management.

4.3 Cyclic Padding

As discussed in Section 4.2, on receiving the *join* message, a client first rebroadcasts the message and then its uplink data. Consider the example network shown in Figure 3. Lets say Alice starts transmitting *join* message at t = Tμs. join message is a 127-symbol PN sequence, so, at 20Mbps, it will take 6.35μs[15] to transmit. Further, due to hardware constraints, it may take Alice up to 2μs more before it can transmit the data packet. Thus, Alice will start transmitting data packet somewhere between $t = T+6.35\mu$s and $t = T + 8.35\mu$s (See Figure 3). AP_1 upon hearing the join message would immediately rebroadcast it. However, due to propagation delay (at most 1 μs [15]) and radio tx-rx turn around (at most 2 μs [15]), it may happen as late as $t = T + 9.35\mu$s. Continuing this computation, we can see that all neighboring clients of AP_2 will start transmitting before $t = T + 27.05\mu$s. Thus, for a given AP, the maximum difference between the start of the data transmission times of its neighboring clients is at most 20.70μs (See Figure 3 where Alice can possibly start transmitting data as early as $t = T + 6.35\mu$s while Don may start as late as at $t = T + 27.05\mu$s). Thus, to ensure that the PN sequences in their data packet headers collide only with the PN sequences of other transmitters, the clients pad cyclic bits after their PN sequences. In Symphony, the length of the cyclic padding is set to 25μs.

By using circular padding and *join* messages broadcasted on wireless, Symphony ensures that it is possible to correlate the PN sequences without requiring global synchronization. By requiring GH to proactively compute the set of suppressed transmissions, Symphony is able to handle unpredictable and non-zero latency on the backbone.

5. HETEROGENEOUS DATA RATES AND PACKET SIZES

In Symphony, different transmitters may select different physical layer data rates due to differences in channel conditions. Since wireless channels are bidirectional in nature[9], the transmitters in Symphony use the received poll packet to estimate the channel to the receiver. This is similar to the Mozart's [2] approach of estimating the physical layer data

join transmission start time	T	[T + 6.35, T + 9.35]	[T + 6.35 + 6.35, T + 9.35 + 9.35] = [T + 12.70, T + 18.70]
data transmission start time	[T + 6.35, T + 8.35]		[T + 12.70 + 6.35, T + 18.70 + 8.35] = [T + 19.05, T + 27.05]

Figure 3: Synchronization: The propagation delay is at most 1μs, the length of 127 symbol PN sequence transmitted at 20Mbps is 6.35μs while the radio tx-rx turn around time is at most 2μs for a total duration of 9.35μs[15]. For a given AP, the maximum difference between the start of the data transmission times of its neighboring clients is at most 20.70μs

rate. However, in Symphony, a client may receive poll from multiple APs. In that case, it estimates the channel with all the APs from whom it received the poll message and chooses the maximum possible data rate such that at least one AP can decode the transmitted packet. Estimating the channel requires the client to estimate the RSS of as many poll packets as possible. However, estimating the RSS of multiple colliding packets is not trivial due to high interference. As explained in Section 2, Symphony borrows the channel estimation technique proposed in Mozart for estimating the RSS of colliding packets.

In Symphony, the channel access time is fixed for all transmitters. A transmitter with better channel may transmit more bits than a transmitter with worse channel. However, it is possible that some transmitter may have good channel but not enough pending data to make complete use of the channel access time. This may lead to wastage of the channel. To minimize the channel wastage, such a transmitter in Symphony will reduce its data rate such that its transmission still fits in the slot size. This reduced data rate sometimes allows the receiver to simultaneously decode two transmissions using Successive Interference Cancellation (SIC) [20]. When using SIC, APs in Symphony decode two packets in the same slot and can suppress up to two transmissions in a single slot.

To implement SIC, the receivers in Symphony need to estimate the RSS of each of the colliding transmitters. For this, we borrow the idea proposed in Mozart[2] whereby PN sequences are used to estimate the RSS. Using SIC in Symphony requires making some changes to the way we construct the dependence graph:

1. In the dependence graph, instead of a vertex being a duplet, it can now be a triplet such that vertex $v_{ijk} = \{P_i, P_j, AP_k\}$ implies that AP_k can decode packets from both P_i and P_j in a single slot using SIC where P_i is the packet with higher RSS at AP_k. Such a vertex is added only if on the basis of estimated RSS and the physical layer data rates used, AP_k determines that it can decode P_i and P_j simultaneously using SIC.

2. A directed edge is drawn from vertex v_{ijk} to another vertex $v_{lmn} = \{P_l, P_m, AP_n\}$ *iff* either P_i interferes at AP_n or P_j interferes at AP_n. Similarly, a directed edge is drawn from v_{ijk} to vertex $v_{ln} = \{P_l, AP_n\}$ *iff* either P_i interferes at AP_n or P_j interferes at AP_n. Finally, a directed edge is drawn from v_{ln} to v_{ijk} *iff* P_l interferes at AP_k.

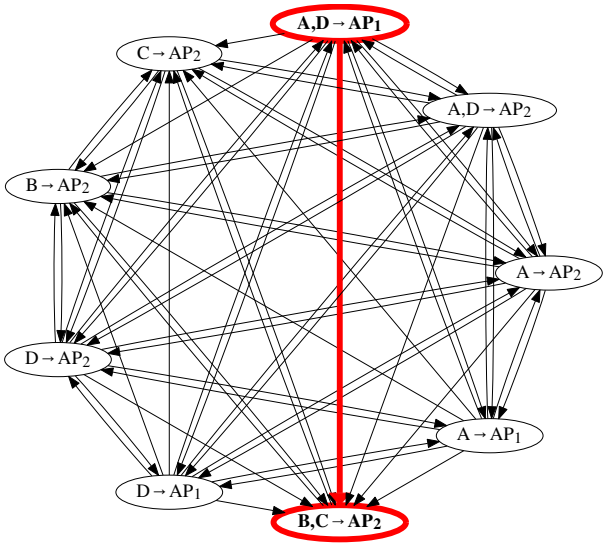


Figure 4: Dependence graph for network topology shown in Figure 1a when using SIC. The chosen acyclic subgraph is also shown in bold.

Figure 5 shows the dependence graph for the network topology shown in Figure 1a when using SIC. As before, to compute the set of suppressed transmissions, the GH picks an Induced Directed Acyclic Subgraph (IDAS) of the dependence graph. If the subgraph has vertices with three tuples, then the corresponding AP needs to suppress two transmitters. To maximize the probability of invoking SIC, Algorithm 1 is modified such that when greedily adding a vertex (Lines 8-15 of Algorithm 1), the GH first only considers the vertices with three tuples. If no such vertex is found, only then the GH adds the vertices with only two tuples.

6. DISCUSSION

In this section, we discuss some issues that need deeper investigation to make our solution more practical.

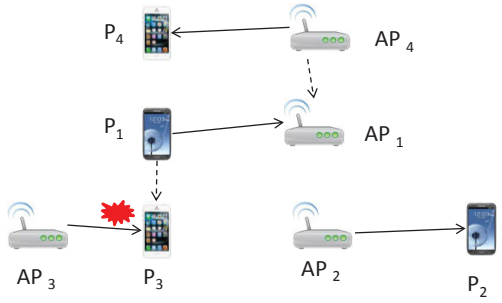


Figure 5: Downlink-uplink coexistence.

1. **Achieving gains on downstream traffic:** In EWLANs, it is generally easier to avoid collisions in downstream data as the APs are aware of which AP has data to send to which client. Multiple algorithms have been proposed in the literature that use this information to schedule downlink transmissions[21, 18, 14]. Symphony, on the other hand, improves performance for upstream data while allowing coexistence of downstream traffic.

In Symphony, due to differences in the distribution of clients, it is possible that for some AP, all its clients are suppressed sooner compared to clients of other APs. Symphony allows these *free* APs to transmit downlink traffic to their clients while other APs keep receiving uplink traffic. For the example network shown in Figure 5, once AP_2, AP_3 and AP_4 become free, they can start transmitting downlink traffic to their neighboring clients. The downlink transmission from AP_2 to P_2 will be successful while the transmission from AP_3 to P_3 will not be successful since it is interfered by P_1's uplink traffic. The transmission from AP_4 to P_4 will also be successful. Although, this downlink transmission interferes with the uplink transmission at AP_1, it is still possible for AP_1 to decode P_1. For this, Symphony requires AP_4 to send its downlink packet to AP_1 on wired backbone. Using that, AP_1 recreates the samples of AP_4's downlink packet and subtracts them from the received samples. Thus, in Symphony, the downlink traffic can coexist with the uplink traffic without causing interference to it.

Once all the APs in a group become free (i.e., they have transmitted *finish*), then the GH schedules interference-free downlink transmissions as discussed before in [14]. Apart from [14, 2], it is also possible to use other techniques proposed in the literature[21, 18] for achieving throughput gains on downlink traffic. Observe that, unlike other algorithms[14, 21], Symphony does not require any knowledge of the network topology. This reduces the overheads in Symphony compared to other algorithms.

2. **Packet subtraction:** In order to decode collided packets, APs in Symphony perform packet subtraction by recreating the samples. To increase the accuracy of subtraction, the APs need to correct for the phase, frequency and sampling offsets[8]. This problem of packet subtraction in presence of interference has been well studied [8, 2]. In our implementation, we use the scheme proposed in ZigZag[8] as it gave the best results.

3. **Sending ACK:** In Symphony, APs exchange the data packets among themselves on the backbone. However, this exchange may take some time due to the wired latency. In order to avoid wasting the channel during the decoding process, the APs immediately send *finish* to the clients without actually decoding the data packets. This allows clients to transmit new data packets when they receive a poll. While the next recovery is going on, the APs exchange the data packets on the wired backbone. Upon successful decoding, the APs send the *acks* to the different transmitters at the end of the next recovery period (during the *wait* period).

4. **Handling decoding errors:** After subtracting a packet, a small amount of *residual noise*[2] is left due to inaccuracies in correcting for the different offsets. If too many packets collide at the same AP, then as the AP cancels packets one-by-one, the residual noise builds up. This may result in an AP being unable to decode the packet of interest due to failed checksum. Further, this may also cause decoding error for the transmissions that are dependent on this transmission. In that case, the AP has two options: (*i*) Different APs may combine frames to correctly decode the received packet [16]. This is particularly useful in dense AP networks where multiple APs (including those who did not suppress any transmitter) may hear the transmission from the same client. (*ii*) If the previous option is not successful, then the APs send *ack* to only the transmitters that were decoded successfully.

5. **Mobility and fading:** It is possible that during recovery, a client may move and its set of neighboring APs may change. Symphony can handle such scenarios as clients in Symphony do not associate with any specific AP. However, if during the recovery, the client moves out of the range of *all* the APs in the group, then this may cause decoding error. In that case, the APs will use the mechanisms discussed above to handle the decoding error. It is also possible that during the recovery, due to fast-fading, some packet (say P_i) is temporarily not received by some of its neighboring APs. This does not cause any decoding error since in Symphony, the set of suppressed transmissions is computed only on the basis of transmissions received in that slot. So, in future when the APs receive the packet again, then it will be suppressed.

7. EXPERIMENTS

In this section, we describe the results from our experiments performed on the GNU radio platform and Universal Software Radio Peripheral (USRP) N210 version 4 radios with WBX daughterboards. In our experiments, we used BPSK with 1/2 convolutional code and the channel frequency was set to 1078 MHz. The phase offset, frequency offset and the sampling offset were cancelled as described in Section 6 while RSS estimation and transmitter identification were done as described in Section 2 and Section 5.

Besides Symphony, we also implemented a version of the IEEE 802.11 protocol. One of the challenges in implementing 802.11 was that USRP has different hardware parameters compared to the commercial 802.11 cards. So, similar to [2], we re-measured the optimum values of all 802.11 parameters (SIFS, DIFS, slot size, ACK timeout) for the N210 hardware. The transmission data rate was kept low (62.5 Kbps) to ensure that the delay between the host computer and the radio was *small* compared to the packet length so that the relative delay values would approximately match[2] the values from off-the-shelf wireless cards.

Apart from implementing our algorithm and IEEE 802.11, we also implemented two other algorithms: (i) **Mozart [2]**;

Table 1: Results from RTT measurements

One-way distance (in hops)	Devices in between	Mean (in μs)	σ (in μs)
2	One switch	117	91.90
4	2 switches + 1 router	829	182.96
5	2 switches + 2 routers	2122	264.31

and, (ii) **Flex Omniscient TDMA**: Here, the clients do not associate with any particular AP but transmit data to and receive data from any of the neighboring APs. The schedule is greedily computed by a central omniscient scheduler. Two links are scheduled at different times *iff* transmitter of one of the link interferes at the receiver of the other link. The knowledge of which nodes have data to send to which other nodes, as well as the complete network topology is provided out-of-band (i.e., without any overhead) to the Flex-TDMA central scheduler and is 100% accurate.

7.1 Results

In our experiments, we placed two USRP APs in two different rooms (as shown in Figure 6a). Four USRP clients were placed at different locations in the three regions so as to create ten different topologies (hidden, non-hidden, mix etc.). In all the topologies, for 802.11 and Mozart, the clients associated to the AP with the strongest signal. Figure 6b shows the CDF of the throughput of clients for the four algorithms. Averaging over all topologies, Symphony (no SIC), Omniscient TDMA, Mozart (no SIC) and IEEE 802.11 provide a per node throughput (in Kbps) of 20.7, 14.4, 11.9 and 7.2, respectively. Thus, Symphony (no SIC) has an average throughput of 43%, 74% and 187% higher than Omniscient TDMA, Mozart (no SIC) and IEEE 802.11, respectively.

7.2 Measurements

Apart from implementing Symphony on the USRP testbed, we also made extensive measurements on the Ohio State University EWLAN. These measurements help us in better designing the algorithm and are also fed to the simulator (See Section 8). The measurements were conducted on a weekday between 10am-4pm (worst case analysis) with the routers and switches also serving the traffic from other desktops and APs on the EWLAN. All the switches and routers on the EWLAN had 1 Gbps interface speed. We measured the round trip time (RTT) between pairs of APs by transmitting and echoing back packets between multiple pairs of APs. The size of the packets was varied from 4-100 bytes to approximate the size of the control packets that would be transmitted from APs to the GH. Table 1 shows the result of the measurements. Using the three-sigma rule, we conclude that even with 5 hops of distance between a pair of APs, with 99.7% confidence, the RTT between the two APs will be less than 2914.93 μs (Mean + 3 times standard deviation).

So, the number of slots for which the GH proactively computes the set of suppressed transmissions (k as discussed in Section 4.2) can be set such that $k \times T \geq 2914.93\mu$s where T is the channel access time (maximum duration of a data packet as discussed in Section 6). In our experiments and simulations, we computed T as $\frac{1500\ bytes}{6\ Mbps} + \delta$ where δ is fixed time required for sending the data packet's header. Value of k was set to 3.

8. SIMULATIONS AND EVALUATIONS

In this section, we explain the results from our ns-3 based trace driven simulations.

8.1 Setup

In our ns-3 based simulations, we randomly placed varying number of APs in a 750m $\times$ 750m field. The number of clients in the network were 5 times the number of APs. Each client associated with the AP from which it received the strongest signal. To generate traffic in the simulator, TCP connections were established between each client and its AP. For this, from the previously collected traffic traces during SIGCOMM[19], we computed the pdf distribution of packet sizes and also the pdf distribution of packet inter-arrival time over all connections. These two pdf distributions were then used to generate both uplink and downlink traffic in the simulation. To create network saturation condition, the number of connections between each client and its associated AP was set to 20.

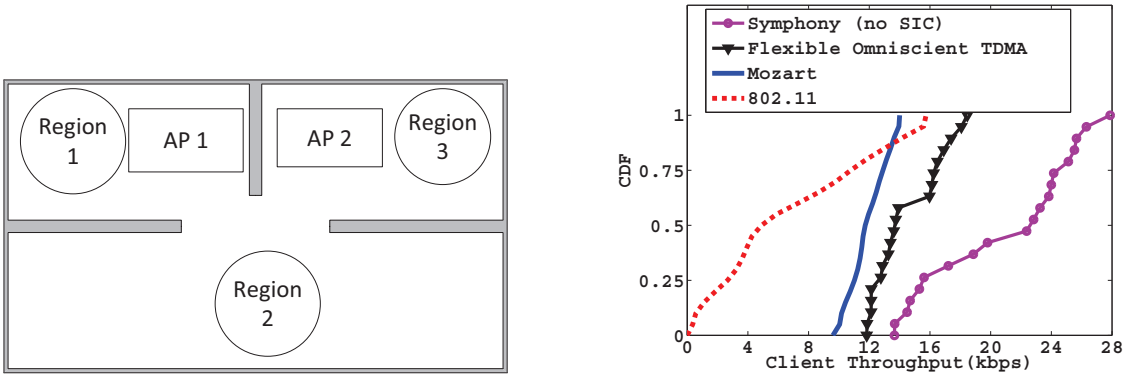
Further, the results from the experiments were fed into the simulator as follows: (i) **PN-Sequence detection and RSS estimation accuracy**: The accuracy of correlating a PN sequence and estimating the transmitter's RSS depends on its SINR [2]. In the simulations, we used the values from [2] to model the PN sequence detection accuracy and the error in RSS estimation; (ii) **Residual noise level**: The level of the noise left after packet subtraction depends on the SINR of the canceled packet[2]. In simulations, we fed the power of the residual noise using the data shown in [2]; and, (iii) **Delays on backbone:** In our simulation, the APs were connected with each other through switched wired ethernet. To every switch, we connected three APs that were geographically close to each other. These switches were then connected to a router. The delay between two APs separated by k hops, was randomly generated using the collected data as discussed in Section 7.2.

To compare the performance of Symphony with state of the art, we implemented various other algorithms: (i) **Mozart** [2]; (ii) **IEEE 802.11 without RTS-CTS**; and, (iii) **Flex Omniscient TDMA** as discussed in Section 7. The channel access time in both Mozart and Symphony was fixed to $\frac{1500\ bytes}{6\ Mbps} + \delta$ where δ is fixed time required for sending the data packet's header. In 802.11, the transmitters follow the data rate adaptation algorithm proposed in [13], while in Mozart and Symphony, the transmitters estimated the channel using the poll packet as described in Section 5.

8.2 Results

8.2.1 Throughput

Figure 7a compares the throughput of different algorithms. On an average, throughput of Symphony was observed to be 63% and 58% higher than TDMA and Mozart, respectively. Further, average throughput of Symphony was 5.6x compared to 802.11. Higher throughput of Symphony can be attributed to two factors: First, cooperative decoding as shown in Figure 1c allows APs in Symphony to simultaneously receive multiple packets. Secondly, when the transmitters have fewer data to send, then it results in throughput loss for TDMA that uses fixed slot length. Although, Symphony also uses fixed slot length, however, by using SIC to decode multiple packets, a single AP in Symphony is able to decode multiple packets simultaneously.



(a) USRP Testbed

(b) CDF plot across different topologies

Figure 6: Experiment testbed and results

To further explore the throughput increase, we also plotted the percentage of data transmissions that were acknowledged by the receiver (See Figure 7b). This number is high for Symphony while for IEEE 802.11, the percentage of packets acknowledged is very low due to hidden terminal problem. Similar to Symphony, the percentage packets acknowledged for TDMA and Mozart were also close to 100%. Symphony had a slightly higher packet acknowledgment rate compared to Mozart since in the case of decoding failure in Symphony, multiple APs exchange frames to cooperatively decode a packet (See Section 6). In 802.11, the packet acknowledgment rate peaks as the number of APs and clients increase and then gradually decreases. This is because with increasing density of APs, it becomes possible for the clients to have stronger channel with the APs, thereby mitigating the effect of hidden terminal (due to data rate adaption and closeness to the AP). However, with increasing density, the probability of two clients picking the same backoff value and the probability of a hidden terminal also increases which decrease the packet acknowledgment rate. Due to these contradicting factors, the packet acknowledgment rate peaks when number of APs is 7.

Figure 7c shows the Jain's fairness index for different algorithms. Symphony has higher fairness since it allows all clients to participate. In other algorithms (including Flex Omniscient TDMA), if a client is in the range of multiple APs, then it may not get a chance to transmit to its AP since the channel in its neighborhood is very likely to be busy.

Figure 7d compares the percentage slots where SIC was used. Symphony uses SIC in more slots than Mozart since in Symphony, the AP can use SIC to suppress any pair of transmitters in its neighborhood. This increases the probability of finding a pair of transmitters that are eligible to be decoded simultaneously. Further, APs in Symphony also cooperate to use SIC to simultaneously transmit two packets to a single client resulting in higher applicability of SIC [20].

8.2.2 *Wired overhead*

In Symphony, APs exchange packets over the wired backbone. Theoretically, the number of unicast packets exchanged on wired backbone can be as many as $\frac{N\times(N-1)}{2}$-times the number of packets decoded on wireless (where N is the number of APs in the network). However, Algorithm 1 minimizes this overhead by reducing the number of edges in the subgraph. Figure 8a shows the overhead as ratio of unicast data sent over wired over the wireless goodput. This ratio increases with increase in number of APs as expected but it plateaus as the number of APs go beyond a certain limit. This is because the broadcast nature of the wireless does not allow every AP in Symphony to suppress a packet. Thus, even with increase in APs, the number of APs that suppress at least one client do not increase, thereby also limiting the number of packets exchanged on the backbone. Further, at any time the ratio is less than 1 implying that for every packet decoded on wireless, at most 1 unicast transmission occurs on the wired backbone.

8.2.3 *Backoff Overhead*

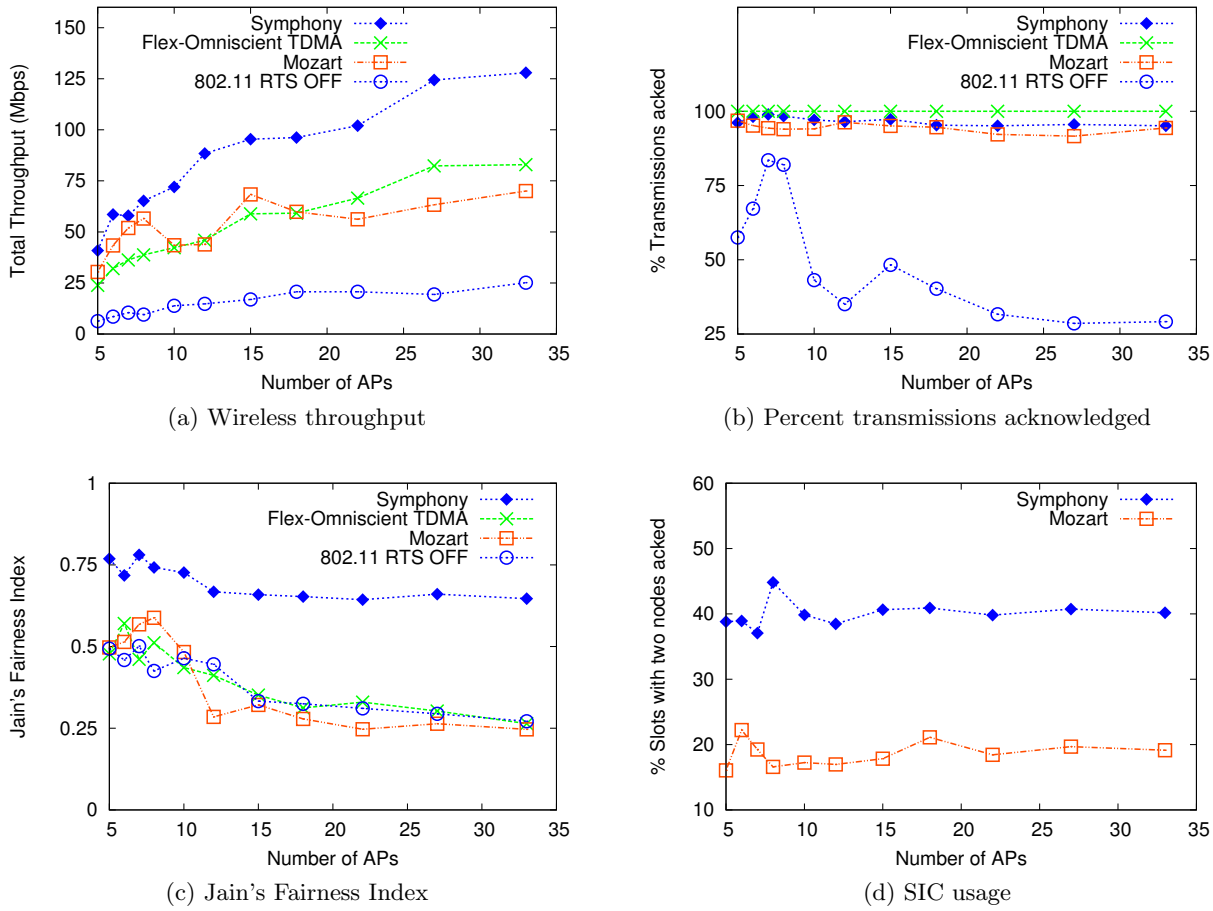
APs in Symphony spend some time in backoff (or timeout) between every recovery. The amount of time spent is given by $\frac{100}{n}$ where n is the number of APs in the group (See Section 4). Figure 8b shows the percent time spend by APs in backoff. APs in Symphony spend less than 1% time in backoff. In Symphony, the backoff overhead is amortized over the length of the recovery period. Further, unlike 802.11, the backoff in Symphony stays constant irrespective of the collision rate.

8.2.4 *Effect of wired ethernet latency*

Due to the unpredictable latency on the wired ethernet, the APs in Symphony proactively compute the schedule for the next 3 slots. To compute the effect of this latency on the optimality of the suppress schedule, we implemented a version of Symphony (called Symphony+) with latency on backbone set to zero. Figure 8c shows that throughput of Symphony+ is within 5% of Symphony indicating that the advance schedule computed by Symphony is close to optimal.

9. RELATED WORK

Backbone usage: The idea of using the wired backbone to increase wireless throughput is not new. In IAC[9], every node is assumed to have at least two antennas. The APs direct the transmitters in the network to align their transmissions such that the APs can cooperatively decode multiple packets. However, unlike [9], Symphony does not assume the



(a) Wireless throughput

(b) Percent transmissions acknowledged

(c) Jain's Fairness Index

(d) SIC usage

Figure 7: Simulation results.

presence of multiple antennas. In MegaMIMO[18], multiple APs cooperatively precode the transmissions such that each client receives only the packets intended for it while the other transmissions cancel-out. However, MegaMIMO requires that transmitters exchange decoded packets among themselves and thus, it works only for the downlink transmissions. On the other hand, Symphony improves the throughput for the uplink traffic.

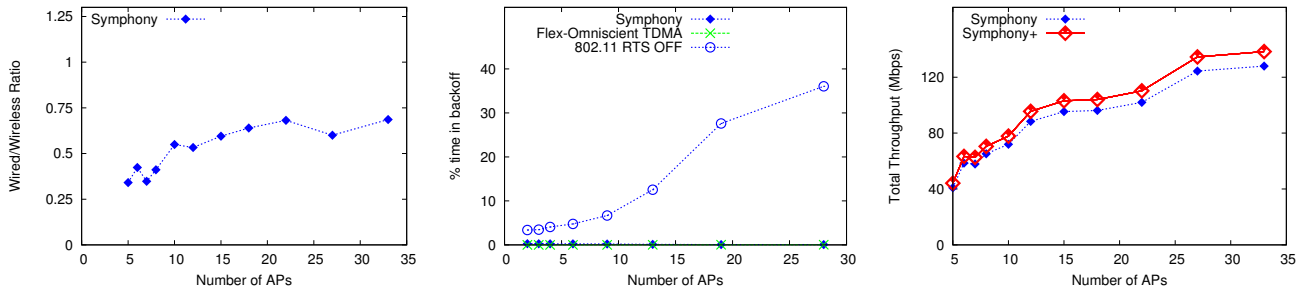
In Epicenter[10], authors propose that APs should exchange coarse representation of symbols to decode corrupted bits. Similarly, authors in [16] also propose that APs exchange bits for decoding corrupted packets while Woo *et al.*[22] propose that APs should exchange raw samples. In all these algorithms, the APs cooperate to decode the same packet whereas in Symphony, APs encourage transmitters to collide and then cooperate to decode multiple packets simultaneously without exchanging the raw samples.

Other MAC algorithms: Algorithms proposed in [17, 4] focus mainly on client-AP association, channel assignment and power control. Authors in [14] propose implementing a backpressure based algorithm for downlink traffic. Similarly, CENTAUR[21] uses a central controller to reduce throughput loss due to hidden terminal and exposed terminal problems. However, their solution is also applicable only to downlink traffic. MiFi [3] also uses centralized coordination of APs and focuses on ensuring fairness. Contrary to the algorithms in [14, 21, 3, 5] that aim to decrease packet collisions, Symphony encourages transmitters to collide and then suppresses the packets so that all the colliding packets can be cooperatively decoded in minimum slots.

Mozart[2] is a recently proposed receiver oriented MAC layer protocol which was discussed in Section 2. In contrast to Mozart, Symphony focuses on leveraging communication in the wired backbone for cooperative decoding of collided packets. This leads to a different set of challenges as discussed in Section 1.

10. CONCLUSIONS

In this paper, we proposed Symphony, a novel backbone-based approach for cooperative packet decoding in Enterprise WLANs. We also designed an algorithm that computes the set of suppressed transmissions. To take into account the practical challenges of high latency, absence of synchronization and a central controller, we proposed a dynamic group based distributed algorithm. To reduce the throughput loss due to variations in packet lengths, we proposed that transmitters reduce their physical layer data rate while receivers use SIC to decode multiple packets simultaneously. Experiments conducted on the USRP testbed show that Symphony provides a throughput of 43% and 187% higher than Omni-



(a) Wired overhead for Symphony. (b) Time spent in backoff (c) Symphony+ has zero wired latency

Figure 8: Simulation results: Overheads.

scient TDMA and IEEE 802.11, respectively. Trace-driven simulations show that, throughput of Symphony is up to 1.63x compared to omniscient TDMA and 5.6x compared to IEEE 802.11.

11. ACKNOWLEDGMENTS

This material is based upon work partially supported by the National Science Foundation under Grant CNS-1161490.

12. REFERENCES

[1] Bansal, T., Chen, B., Sinha, P., and Srinivasan, K. Symphony. Tech. rep., 2013. `http://www.cse.ohio-state.edu/~bansal/symphonytechrep.pdf`.

[2] Bansal, T., Chen, B., Srinivasan, K., and Sinha, P. Mozart: Orchestrating Collisions in Wireless Networks. Tech. rep., 2013. `http://www.cse.ohio-state.edu/~bansal/mozart.pdf`.

[3] Bejerano, Y., and Bhatia, R. S. MiFi: A Framework for Fairness and QoS Assurance for Current IEEE 802.11 Networks with Multiple Access Points. *IEEE/ACM Transactions on Networking (TON) 14*, 4 (2006), 849–862.

[4] Bejerano, Y., Han, S.-J., and Li, L. E. Fairness and Load Balancing in Wireless LANs Using Association Control. In *Proc. ACM MOBICOM* (2004).

[5] Cidon, A., Nagaraj, K., Katti, S., and Viswanath, P. Flashback: Decoupled Lightweight Wireless Control. In *Proc. ACM SIGCOMM* (2012).

[6] Festa, P., Pardalos, P. M., and Resende, M. G. Feedback Set Problems. *Handbook of Combinatorial Optimization 4* (1999), 209–258.

[7] Gold, R. Optimal Binary Sequences for Spread Spectrum Multiplexing. *Information Theory, IEEE Transactions on 13*, 4 (1967), 619 –621.

[8] Gollakota, S., and Katabi, D. Zigzag Decoding: Combating Hidden Terminals in Wireless Networks. In *Proc. ACM SIGCOMM* (2008).

[9] Gollakota, S., Perli, S. D., and Katabi, D. Interference Alignment and Cancellation. In *Proc. ACM SIGCOMM* (2009).

[10] Gowda, M., Sen, S., Roy Choudhury, R., and Lee, S. Cooperative Packet Recovery in Enterprise WLANs. In *Proc. IEEE INFOCOM* (2013).

[11] Johannessen, S. Time Synchronization in a Local Area Network. *Control Systems, IEEE 24*, 2 (2004).

[12] Koutsonikolas, D., Salonidis, T., Lundgren, H., LeGuyadec, P., Hu, Y. C., and Sheriff, I. TDM MAC Protocol Design and Implementation for Wireless Mesh Networks. In *Proc. ACM CoNEXT* (2008).

[13] Lacage, M., Manshaei, M. H., and Turletti, T. IEEE 802.11 Rate Adaptation: A Practical Approach. In *Proc. ACM MSWIM* (2004).

[14] Laufer, R., Salonidis, T., Lundgren, H., and Le Guyadec, P. XPRESS: A Cross-Layer Backpressure Architecture for Wireless Multi-Hop Networks. In *Proc. ACM MOBICOM* (2011).

[15] Magistretti, E., Gurewitz, O., and Knightly, E. 802.11 ec: Collision Avoidance Without Control Messages. In *Proc. ACM MOBICOM* (2012).

[16] Miu, A., Balakrishnan, H., and Koksal, C. E. Improving Loss Resilience with Multi-Radio Diversity in Wireless Networks. In *Proc. ACM MOBICOM* (2005).

[17] Murty, R., Padhye, J., Chandra, R., Wolman, A., and Zill, B. Designing High Performance Enterprise Wi-Fi Networks. In *Proc. USENIX NSDI* (2008).

[18] Rahul, H., Kumar, S., and Katabi, D. MegaMIMO: Scaling Wireless Capacity with User Demand. In *Proc. ACM SIGCOMM* (2012).

[19] Schulman, A., Levin, D., and Spring, N. CRAWDAD data set umd/sigcomm2008. Downloaded from http://crawdad.cs.dartmouth.edu/umd/sigcomm2008.

[20] Sen, S., Santhapuri, N., Choudhury, R. R., and Nelakuditi, S. Successive Interference Cancellation: Carving Out MAC Layer Opportunities. *IEEE Trans. Mob. Comput. 12*, 2 (2013), 346–357.

[21] Shrivastava, V., Ahmed, N., Rayanchu, S., Banerjee, S., Keshav, S., Papagiannaki, K., and Mishra, A. CENTAUR: Realizing the Full Potential of Centralized WLANs Through a Hybrid Data Path. In *Proc. ACM MOBICOM* (2009).

[22] Woo, G. R., Kheradpour, P., Shen, D., and Katabi, D. Beyond the Bits: Cooperative Packet Recovery Using Physical Layer Information. In *Proc. ACM MOBICOM* (2007).