XPRESS: A Cross-Layer Backpressure Architecture for Wireless Multi-Hop Networks

Rafael Laufer
UCLA
Los Angeles, USA

Theodoros Salonidis
Technicolor
Paris, France

Henrik Lundgren
Technicolor
Paris, France

Pascal Le Guyadec
Technicolor
Paris, France

ABSTRACT

Contemporary wireless multi-hop networks operate much below their capacity due to the poor coordination among transmitting nodes. In this paper we present XPRESS, a cross-layer backpressure architecture designed to reach the full capacity of wireless multi-hop networks. Instead of a collection of poorly coordinated wireless routers, XPRESS turns a mesh network into a wireless switch. Transmissions over the network are scheduled using a throughput-optimal backpressure algorithm. Realizing this theoretical concept entails several challenges, which we identify and address with a cross-layer design and implementation on top of our wireless hardware platform. In contrast to previous work, we implement and evaluate backpressure scheduling over a TDMA MAC protocol, as it was originally proposed in theory. Our experiments in an indoor testbed show that XPRESS can yield up to 128% throughput gains over 802.11.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*Wireless communication*

General Terms

Algorithms, Design, Experimentation

Keywords

Backpressure scheduling and routing

1. INTRODUCTION

Existing networks are designed in layers, where protocols operate independently at each layer of the network stack. This approach provides flexibility with a modular design and standardization, but it may result in severe performance degradation when these protocols do not cooperate well. This is usually the case of wireless multi-hop networks, where noise and interference at lower layers affect the routing and congestion control performed at upper layers.

A common approach to address these performance issues is then to modify a single layer of the protocol stack, while keeping other layers intact. Cross-layer architectures offer a radical alternative by advocating cooperation among the multiple layers of the protocol stack. At the core of these architectures is the backpressure scheduling algorithm [24], which, in theory, achieves the network capacity.

Translating this theoretical concept into a practical system, however, entails several challenges, mainly due to its idealized assumptions. In essence, backpressure assumes a globally synchronized time-slotted MAC protocol as well as a central controller that computes and disseminates a schedule (i.e., a set of links allowed to transmit) for each time slot. Moreover, the schedule computation requires the global knowledge of per-flow queue backlogs and network state (i.e., link quality and link interference pattern), which therefore must be measured at the wireless nodes and provided to controller in a timely manner. Recently, practical backpressure systems that relax some of these assumptions and approximate the backpressure algorithm on top of the 802.11 MAC protocol have been proposed [2, 20, 27]. Even though these approaches have shown performance benefits, the step towards a practical system implementing optimal backpressure scheduling has not yet been made.

In this paper we present XPRESS, a throughput-optimal backpressure architecture for wireless multi-hop networks. In XPRESS, a mesh network is transformed into a wireless switch, where packet routing and scheduling decisions are made by a backpressure scheduler. XPRESS is composed of a central controller, which performs backpressure scheduling based on the measured wireless network state, and also of the wireless nodes, which periodically provide the network measurements and execute the computed schedule using a cross-layer protocol stack. The implementation of XPRESS on our 802.11 platform resulted in novel techniques that overcome the above-mentioned backpressure challenges and have a wider applicability to the design of centralized multi-hop wireless systems. Our contributions are as follows.

First, the XPRESS cross-layer stack gracefully integrates the transport, network, and MAC layers. In order to achieve synergy among these layers on our programmable 802.11 platform, we had to implement (a) a congestion control scheme to ensure the scheduler operates within the capacity region; (b) a coordination mechanism between network-layer flow queues and MAC-layer link queues, which enables per-link queue implementation on memory-constrained wireless interfaces; and (c) a multi-hop TDMA MAC protocol that not only ensures global synchronization among nodes, but also enables coordinated transmissions within slot boundaries according to the exact backpressure schedule.

Second, we find that a TDMA MAC on top of 802.11 PHY results in binary interference patterns. This relaxes the channel state estimation requirement from exact values to binary state (i.e., delivery ratios are close to 0% or 100%), which allows us to efficiently find the link transmission sets and accurately estimate the queue backlogs of the network. Complementary techniques, such as multi-slot schedule computations and speculative scheduling, reduce the protocol overhead further and provide the scheduler at the controller with a longer computation time budget at the expense of outdated, but still accurate, network state.

Third, we propose an interference estimation mechanism with only $O(N)$ measurement complexity that allows the backpressure scheduler to determine at the TDMA frame time scale the links which can transmit without interference. The mechanism uses received signal strength (RSS) as well as an adaptive technique based on packet loss to cope with the RSS measurement limitations of 802.11.

Fourth, our evaluations in an indoor testbed show that XPRESS provides close to perfect fairness in small-scale centralized WLANs and 63%–128% throughput gains over 802.11 in multi-hop mesh configurations. We also show that XPRESS accurately emulates the optimal backpressure schedule and delivers relatively low delays when operating close to capacity.

Finally, we provide an analysis of the communication and computation overhead of XPRESS and identify different system design choices and limitations.

2. BACKPRESSURE SCHEDULING

The backpressure algorithm was introduced in [24] as a scheduling policy that maximizes the throughput of wireless multi-hop networks. Assuming slotted time, the basic idea of backpressure scheduling is to select the "best" set of non-interfering links for transmission at each slot. We now describe this idea in a 4-node network with two flows, black and gray, from node A to D, depicted in Figure 1. Each node maintains a separate queue for each flow. For each queue, the number of backlogged packets is shown. Assume that we have two link sets, $\{(A, B), (C, D)\}$ and $\{(A, C), (B, D)\}$, shown as continuous and dashed lines, respectively. The links in each set do not interfere and can transmit in the same time slot.

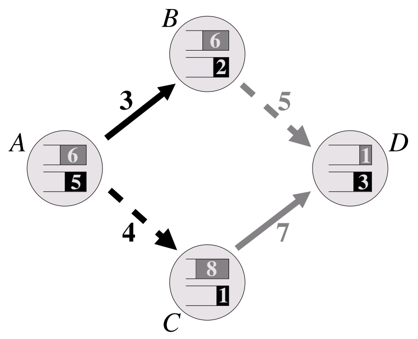


Figure 1: Backpressure scheduling in a network with two flows, black and gray, from A to D. Links in sets $\{(A, B), (C, D)\}$ (continuous) and $\{(A, C), (B, D)\}$ (dashed) can be scheduled in the same slot.

The scheduler executes the following three steps at each slot. First, for each link, it finds the flow with the maximum differential queue backlog. For example, for link (A, B), the gray flow has a difference of 0 packets and the black flow has a difference of 3 packets. The maximum value is then assigned as the weight of the link (see Figure 1). Second, the scheduler selects the set of non-interfering links with the maximum sum of weights for transmission. This requires to compute the sum of link weights for each possible set. In the example, set $\{(A, B), (C, D)\}$ sums to $3 + 7 = 10$ and set $\{(A, C), (B, D)\}$ sums to $4 + 5 = 9$. The scheduler then selects the set with the maximum sum of weights, i.e., $\{(A, B), (C, D)\}$, to transmit at this slot. Finally, packets from the selected flows are transmitted on the selected links, i.e., black flow on link (A, B) and gray flow on link (C, D). The same computation is then performed at every slot.

2.1 Backpressure Algorithm

More formally, the backpressure scheduling algorithm consists of the following steps executed for each time slot.

Flow scheduling and routing: For each link (i, j), select the flow f^*_{ij} with the maximum queue differential backlog

$$f^*_{ij} = \arg\max_{f \in F} \; (q^f_i - q^f_j), \tag{1}$$

where q^f_i and q^f_j are the queue backlogs for flow f at nodes i and j, respectively, and F is the set of flows. Equation (1) implicitly performs routing by selecting the link (i, j) that each flow may use during the slot. The weight w_{ij} of each link is then selected as the weight of flow f^*_{ij}:

$$w_{ij} = \max_{f \in F} \; (q^f_i - q^f_j). \tag{2}$$

Link scheduling: Select the optimal link capacity vector $\boldsymbol{\mu}^* = (\mu^*_{ij})$ that satisfies

$$\boldsymbol{\mu}^* = \arg\max_{\boldsymbol{\mu} \in \Lambda} \; \sum_{(i,j)} \mu_{ij} w_{ij}, \tag{3}$$

where $\boldsymbol{\mu} = (\mu_{ij})$ are the link capacity vectors. The capacity μ_{ij} of each link (i, j) is the maximum rate in bits/s that the link can transmit subject to the channel state and the interference due to the other links in the vector. The set of all feasible link capacity vectors define the capacity region Λ.

Transmission: During the time slot, a selected link (i, j) transmits a packet from flow f^*_{ij} using rate μ^*_{ij}.

2.2 Congestion Control

The backpressure algorithm is throughput-optimal when the flow rates are within the capacity region. This issue can be addressed by combining the backpressure algorithm with the network utility maximization (NUM) framework, originally proposed for wireline networks [8]. This framework leads to a simple distributed congestion control algorithm where the source s of each flow f adjusts the flow rate x_f as

$$x^*_f = \arg\max_{x_f \geq 0} \; U_f(x_f) - x_f q^f_s = U'^{-1}_f(q^f_s), \tag{4}$$

where q^f_s is the queue backlog for flow f at the source s and $U'^{-1}_f(q^f_s)$ is the inverse of the first derivative of the utility function $U_f(.)$ at the point q^f_s. In [4, 6, 11], it is proven that the congestion control scheme of Equation (4) regulates the flow rates to be within the capacity region and cooperates with the backpressure scheduler to maximize throughput.

3. XPRESS DESIGN

This section presents the XPRESS system, a cross-layer backpressure architecture for wireless multi-hop networks. To our knowledge, XPRESS is the first system to implement backpressure scheduling over a time-slotted MAC, as it was originally proposed in theory. We first provide a high-level system overview and then we detail the data plane and control plane designs. Finally, we describe the design of our backpressure scheduler with speculative scheduling.

3.1 Overview

In XPRESS the wireless network is composed of several mesh access points (MAPs), a few gateways (GWs), and a mesh controller (MC), as depicted in Figure 2. We use the term "node" to refer to a mesh node that can be either a MAP or a GW. The MAPs provide wireless connectivity to mobile clients and also operate as wireless routers, interconnecting with each other in a multi-hop fashion to forward user traffic. Mobile clients communicate with MAPs over a different channel, and thus are not required to run the XPRESS protocol stack. The GWs are connected to both the wireless network and the wired infrastructure, and provide a bridge between the two. The MC is responsible for the coordination of the wireless transmissions in the network, and it is analogous to a switching control module. In our design, the MC is deployed in a dedicated node in the wired infrastructure and connects to the gateways through high-speed links. In an alternative design, the MC could be implemented within one of the gateways, if necessary.

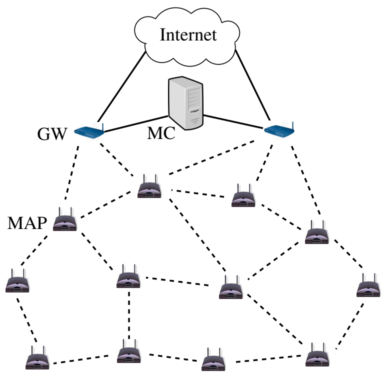


Figure 2: The XPRESS architecture, composed of mesh access points, gateways, and a mesh controller.

At a high level, the operation of XPRESS is described as follows. XPRESS runs a slotted MAC protocol, where a sequence of slots are organized into frames. For each slot, XPRESS selects a set of non-interfering links to transmit based on the flow queue lengths and the network state. Each node thus maintains per-flow queues, and monitors adjacent links to estimate interference and losses. The queue lengths and network monitoring results are periodically transmitted to the MC over an uplink control channel. Upon reception of this information, the MC updates its local topology and interference databases, and runs the backpressure scheduler to calculate the throughput-optimal schedule for multiple upcoming slots (i.e., a frame). The MC then disseminates the resulting schedule to the nodes over a downlink control channel. The nodes in turn apply the new schedule for transmissions in the next frame. This cycle repeats periodically.

3.2 Data Plane

The XPRESS data plane spans across the transport, network, and MAC layers of the protocol stack, as depicted in Figure 3. The transport and network layers implement congestion control and flow scheduling, respectively. The MAC layer implements link scheduling and a TDMA MAC protocol. The organization of these modules into host OS kernel and network interface card firmware depends on the architecture used. For convenience, Figure 3 shows this organization on our testbed devices (cf. Section 6), where the full MAC firmware resides on the wireless cards while the upper layers reside in the host OS kernel. In the figure, diamonds represent packet classifiers, while circles represent packet schedulers. The data flow from left to right are outgoing packets originating from the applications to the wireless medium; the data flow in the opposite direction are incoming packets that are routed or delivered to the applications. Packets in the slotted wireless medium (far right), which are neither incoming nor outgoing, represent transmissions between two other nodes in the network.

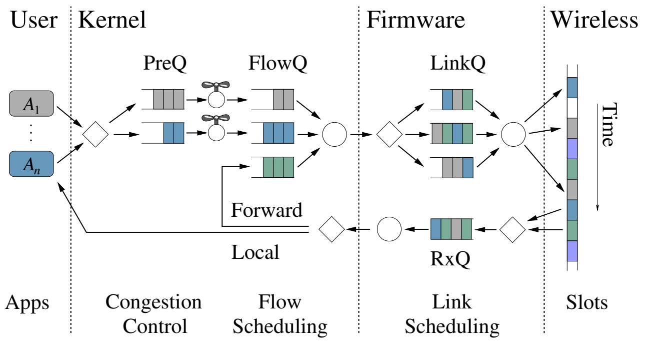


Figure 3: The data plane at XPRESS nodes. Diamonds are packet classifiers while circles are packet schedulers. Rate control and flow scheduling occur at the kernel; link scheduling occurs at the firmware.

XPRESS flows: In XPRESS, a flow is defined at the IP layer by its source and destination mesh nodes. Our design is general and can easily accommodate other flow definitions. However, compared to the usual 5-tuple flow definition of source and destination IP addresses, source and destination transport ports, and transport protocol (i.e., TCP or UDP), this design decision reduces processing and communication overhead in XPRESS at the expense of flow granularity.

Congestion control and flow scheduling: Locally originated packets first pass through a flow classifier, represented by the left-most diamond in Figure 3. Each flow has two individual queues, namely, a pre-queue (PreQ) and a flow queue (FlowQ). After classification, packets are inserted into the PreQ and must pass through the congestion controller, represented by the faucet handle in the figure. Congestion control is performed according to Equation (4) and depends only on the length of the local FlowQ. A longer FlowQ reduces the allowed input rate, while a shorter FlowQ allows a higher rate. After congestion control, packets enter the FlowQ and wait to be scheduled. The kernel-space packet scheduler is synchronized with the slotted MAC with respect to time and link queue state. Just shortly before a scheduled transmission slot starts, the kernel scheduler dequeues a packet from the scheduled FlowQ and sends it down to the firmware link queue for transmission.

Link scheduling: The MAC protocol keeps an individual queue for each neighbor in order to enable *link scheduling*, which allows a higher spatial reuse than *node scheduling*. As packets dequeued from the FlowQ arrive at the link-level packet classifier, they are classified according to the destination MAC address and inserted into the appropriate link queues (LinkQ). The slotted MAC, realized by a TDMA MAC protocol (cf. Section 4.3), maintains network-wide node synchronization, and ensures that transmissions occur strictly within slot boundaries. When a transmission slot starts, the MAC protocol dequeues a packet from the scheduled LinkQ and transmits it over the air. If the transmission fails and the retransmission limit is not reached, the packet remains in the appropriate LinkQ until the next slot for the same neighbor.

Packet reception and forwarding: Once a packet is received, it is first filtered based on the destination MAC address and then inserted into a single receive queue (RxQ) at the firmware. The packet is delivered to the network layer at the kernel, where it is routed and tagged for local delivery or forwarding. In the latter case, the packet is inserted into the respective FlowQ and waits to be scheduled, just like a locally-generated packet after passing congestion control.

3.3 Control Plane

The different system components must exchange control information to coordinate the network-wide transmissions. Figure 4 depicts the XPRESS control plane with its different components at the mesh controller (MC) and a mesh access point (MAP). The uplink control channel is represented by the arrows labeled from 1 to 3, while the downlink control channel is represented by arrows 4 and 5. There is also an internal control channel (dashed arrow) between the MAC and the the kernel packet scheduler, to synchronize them with respect to time and link queue (LinkQ) state.

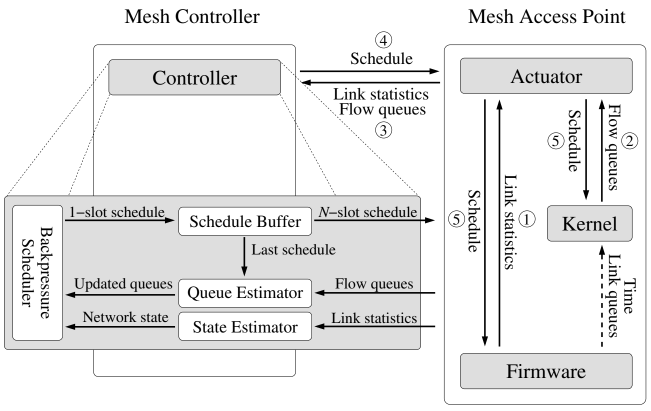


Figure 4: The XPRESS control plane and its system components. Numbered arrows indicate messages exchanged every frame, whereas the dashed arrow represents an internal message exchanged every slot between the firmware and the kernel of the MAP.

Uplink channel: Each node runs an actuator application that communicates with the controller application at the MC. A schedule computation cycle begins at the start of a new frame. At this point, the TDMA MAC protocol notifies the actuator about the new frame and piggybacks in the same message the link statistics collected during the last frame (e.g., received signal strength and delivery ratio). This is the Step 1 in Figure 4. The actuator then collects the FlowQ lengths (Step 2), combines all information and sends it to the MC using the uplink channel (Step 3).

Mesh controller (MC): The zoom of the MC in Figure 4 shows the different steps taken for the calculation of a TDMA frame schedule composed of N slots. First, the link statistics are used to estimate the network state, namely, interference relations and link loss rates. We explain this step in Section 5. The FlowQ lengths are also collected and fed into a backpressure scheduler. The scheduler then uses the FlowQ lengths and network state to compute the network schedule, using the backpressure algorithm (cf. Section 2) and our speculative scheduling technique, which we detail in Section 3.4.

Downlink channel: When the schedule computation is finished, the MC disseminates the new schedule using the downlink control channel (Step 4). The actuator receives this packet and forwards the new schedule both to the OS kernel as well as to the MAC layer (Step 5). The TDMA MAC starts using this new schedule for data transmission in the next frame. Packets will then be dequeued from the FlowQs to the LinkQs in accordance with this new schedule.

3.4 Backpressure Scheduler

XPRESS introduces a speculative scheduling technique to reduce scheduling overhead. This technique computes a schedule for a group of slots on a TDMA frame basis and performs the optimal backpressure computation for all slots in the frame based on speculated network queue state.

Figure 5 depicts the speculative scheduling operation. At the start of TDMA frame k, the MC computes a schedule $S(k+1)$ for all data slots of the *next* frame $k+1$. This approach provides the MC with a time budget of one TDMA frame to perform optimal computations. However, it comes at the expense of uncertainty due to changes in the queue backlogs during frame k. These changes are due to incoming packets at the source node of each flow, wireless losses, and the schedule $S(k)$ executed by the nodes during the slots of frame k. The MC addresses this uncertainty using the schedule $S(k)$ it computed during the previous frame $k-1$, as well as the FlowQ lengths $Q(k)$ and source rate estimates $R(k)$ provided by the mesh nodes at the start of frame k. Additionally, as explained in Section 5, the scheduler uses only links of high packet delivery ratios which reduces the uncertainty of wireless losses.

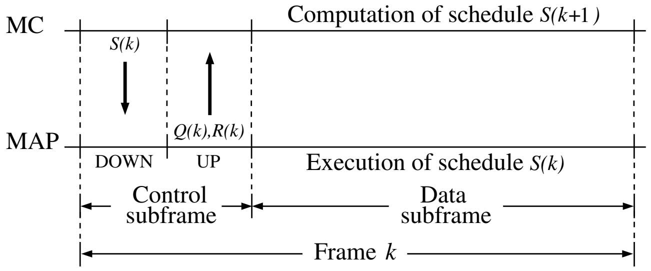


Figure 5: Queue sampling and schedule computation in XPRESS. FlowQ lengths $Q(k)$ and rates $R(k)$ provided at the start of frame k are used to estimate $Q(k+1)$ and compute the schedule $S(k+1)$ for the next frame $k+1$.

First, the MC computes an estimate $\hat{Q}(k+1)$ of the queue backlogs at the start of frame $k+1$. The MC updates $Q(k)$ by adding $R(k)$ to the flow queue of each source node as an estimate of the number of incoming packets at frame k. Then, the MC locally emulates the transmissions of schedule $S(k)$ on this updated queue state, that is, for each slot, the scheduled FlowQs of transmitters are decremented and the FlowQs of receivers are incremented until $\hat{Q}(k+1)$ is obtained.

Given $\hat{Q}(k+1)$, the MC computes the schedule of the first slot of frame $k+1$ with the backpressure algorithm (Section 2). After this computation, the MC updates the scheduled FlowQs, creating a new queue estimate for the second slot (see Figure 4). The rate $R(k)$ is also used to update the queue estimate of each slot according to the input rate at each source. With the new queue estimate, the backpressure algorithm computation is repeated. This process continues for the following slots, until the schedule $S(k+1)$ of all data slots of frame $k+1$ are calculated. After the computation, the MC transmits this schedule to the mesh nodes for execution during frame $k+1$. In Section 6.4 we experimentally validate our speculative scheduling technique and show that the estimated queue backlog $\hat{Q}$ closely follows the actual backlog Q.

4. XPRESS IMPLEMENTATION

The XPRESS design is general and can be realized on a wide range of platforms. In this section, we describe the main components of our cross-layer implementation in the Linux OS and the firmware of our WiFi cards. We follow a top-down approach and describe these components in the order of the outgoing data path in Figure 3.

4.1 Congestion Control

Congestion control is performed only at the source node of each flow by adjusting the flow input rate in accordance with Equation (4). More precisely, the source rate x_f of each flow f is continuously adjusted to the optimal rate $x_f^* = U_f'^{-1}(q_s^f)$ for the flows to remain within the capacity region. In XPRESS, we use $U_f(x_f) = K\log(x_f)$ as the utility function, where K is a constant parameter defined later in Section 6. The logarithmic function allows a good tradeoff between fairness and efficiency in wireless networks [19]. The maximum allowed rate of each flow is then periodically adjusted to $x_f^* = K/q_s^f$, where q_s^f is the length of the FlowQ of flow f at the source s.

We implement this congestion control in the Linux kernel between the PreQ and FlowQ. Locally generated outgoing packets are intercepted using the Linux Netfilter local-out hook. These packets are classified by flow and put into their respective PreQ. A per-flow token-bucket rate controller adjusts the rate at which they are allowed to enter the FlowQ. This rate control is performed by periodically inserting tokens into the bucket according to the optimal rate x_f^*. When there are enough tokens in the bucket, a packet is dequeued from the PreQ and sent to the FlowQ.

4.2 Queues and Scheduler

Flow queues: Outgoing packets are intercepted using the Netfilter post-routing hook in the Linux kernel. Intercepted incoming packets that have been routed, and thus are ready to be forwarded, are classified and put into the corresponding FlowQ. We pass the FlowQ backlog information to the actuator module through the Linux `/proc` interface. The actuator in turn forwards this information over the uplink control channel to the MC for schedule computation.

Backpressure scheduler: In our speculative scheduling, the schedule for each slot is computed at the MC using the backpressure algorithm (cf. Section 2). In Section 5, we show that this can be reduced to a maximum weight independent set (MWIS) computation in our system due to binary interference. The MWIS computation is an NP-hard problem in theory and can be computationally intensive in practice. Our C++ implementation is based on an algorithm for enumerating maximal independent sets at the beginning of each frame [15]. We then find the MWIS using a linear search over the independent sets. For efficiency, these sets could be stored in a heap structure keyed by their weights. At each slot, queue lengths change, which triggers a heap update. After the update, the new MWIS can be found as the root of the heap.

The MC runs on a Linux PC with a quad-core 2.7 GHz AMD Opteron® computer, with 16 GB RAM. In our testbed the execution time of our MWIS implementation on the MC never exceeded 100 μs. In Section 7 we show that the MWIS problem can be solved efficiently for practical network sizes.

Kernel packet scheduler: The packet scheduler, in turn, is implemented in the MAPs as a kernel thread that waits for the computed schedule. The schedule contains information about which flows to transmit in each slot, as well as the next hop to be used by each flow. Based on this schedule and the instantaneous LinkQ lengths (see below), the packet scheduler decides when to dequeue packets from the FlowQs and send them to the LinkQs.

Link queues: We implement LinkQs as individual neighbor queues at the firmware level to allow link scheduling. Individual neighbor queues are needed in link scheduling to prevent head-of-line blocking caused by a single common transmission queue. This problem may occur if we have wireless losses and the packet at the head of the queue is destined for a different neighbor than the one assigned for the slot, resulting in no packet transmitted during that slot.

Cross-layer queue synchronization: Implementing link scheduling at the firmware is challenging due to the limited memory of wireless cards. As an example, our platform can only hold about a dozen full-size packets in the firmware at any time. As a result, the kernel and the firmware must be tightly synchronized with respect to both memory utilization and time to avoid *memory-exhaustion blocking* and slot under-utilization. Memory-exhaustion blocking occurs if a given LinkQ has accumulated too many packets, leaving no memory space for other queues. Once the firmware memory is full, no more packets can be sent from the FlowQs to *any* LinkQ. At the same time, the kernel must send a sufficient number of packets to populate the different LinkQs of the upcoming slots. In this case, transmission slots will remain unused if the scheduled LinkQ is empty.

We implement a cross-layer queue synchronization to address these issues. The firmware periodically advertises its LinkQ lengths as well as the current time slot to the kernel. In our implementation, this occurs at every slot for a tight synchronization. The kernel uses this information to send each packet to the firmware just two slots before its actual transmission, unless the corresponding LinkQ already has a sufficient number of packets in its queue.

4.3 TDMA MAC Protocol

We implement a time-slotted MAC protocol in the firmware of our 802.11a/b/g cards and disable inherent CSMA functionality, including carrier sensing, backoff, RTS/CTS and NAV. This gives us full control of transmission times and allow us to divide the MAC operation into frames and time slots. In our implementation, each TDMA frame is composed of a group of slots, which are divided into a control subframe (CS) followed by a data subframe (DS). We adopt our multi-hop TDMA slot synchronization protocol in [9] to synchronize MAPs over the wireless channel during the CS. After the control slots, nodes are synchronized and the rest of the frame is composed of data slots. Each data slot is assigned to a particular set of non-interfering links based on the schedule computed by the MC. The MAC packet scheduler receives this schedule from the actuator, and then takes care of transmitting packets according to the schedule and within slot boundaries. The actuator notifies the MAC about the new schedule as soon as it receives it from the MC. In our case, a new schedule is applied on every frame.

5. INTERFERENCE ESTIMATION

We now introduce the design of our interference estimation technique to provide the backpressure scheduler with the link transmission sets and the corresponding capacities of their links. The capacity μ_{ij} of each link (i,j) is estimated on a TDMA frame time scale as $P_{ij} \times R_{ij}$ where P_{ij} is the packet delivery ratio (PDR) and R_{ij} is the PHY rate of link (i,j) during the TDMA frame.

Finding the link transmission sets and their capacities is a challenge because each link capacity depends on both the channel condition and the interference created by the other links in the set. A direct approach would be to enumerate and schedule each link set in the same slot, and then measure the PDR of their links. In a network with L links and N_r PHY rates, this requires $O(N_r \times 2^L)$ measurements during each TDMA frame, which is prohibitive.

In order to understand how interference manifests in our TDMA system, we perform the following experiment for all link pairs that do not share a node in our testbed[1]. The links of each pair simultaneously transmit backlogged UDP traffic for 1 minute using broadcast packets at 24 Mbps PHY rate. During this time, the receivers measure the received signal strength (RSS) and PDR values at each TDMA frame. Figure 6 depicts the CDF of the measured PDRs. We observe that interference creates a binary pattern where only 10% of links have intermediate PDR values between 0% and 90%. The dominance of the binary pattern is due to the clock synchronization of the TDMA protocol and the way 802.11a MAC decodes packets. According to our measurements, the TDMA protocol keeps the node clocks synchronized within 4 μs. Thus, interfering transmissions overlap with packet reception at the very beginning of the 802.11a MAC preamble, where capture effect is weak [10]. This is independent of PHY data rate because it occurs at the beginning of the MAC preamble, where the lowest PHY data rate is used.

The binary interference phenomenon has important implications on our system. First, it is enough to estimate "low" or "high" PDR rather than exact values. Second, the link transmission sets consist only of links with "high" PDR

[1]In our current implementation, nodes have a single radio; therefore, links that share a common node always interfere.

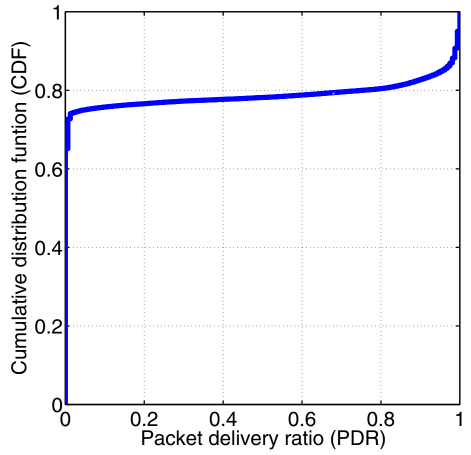


Figure 6: The cumulative distribution function (CDF) of the TDMA frame PDR of links transmitting backlogged in pairs at 24 Mbps using the TDMA MAC protocol.

and can be identified as the independent sets of a binary conflict graph. This bypasses exhaustive enumeration and measurement on all 2^L link combinations to determine the link transmission sets. Finally, the link scheduling problem of the backpressure algorithm, i.e., Equation (3), reduces from an exhaustive search over all link capacity vectors to a maximum weight independent set (MWIS) problem in the conflict graph.

5.1 Interference Measurement

Nodes in XPRESS measure the RSS of each neighbor from the synchronization beacons they transmit in the control subframe of each TDMA frame. Each node has an assigned slot in this subframe, at which the node transmits its beacon using the lowest 802.11a PHY rate (i.e., 6 Mbps) without interference. This results in one RSS measurement per frame for each link. Based on these measurements, we estimate the signal-to-interference ratio (SIR) of a link (i,j) under the interference of a transmitting node k as the difference $S_{ij} - S_{kj}$, between S_{ij}, the RSS of link (i,j), and S_{kj}, the RSS of link (k,j), both in dBm. If the SIR of the link exceeds a threshold, which depends on the PHY rate, the link PDR is estimated "high." Since RSS values are independent of PHY rate, this approach can estimate the PDR of all transmission sets for all PHY rates and reduce the number of measurement complexity from $O(N_R \times 2^L)$ to $O(N)$.

Despite its advantages, this RSS approach can be far from perfect for two reasons. First, the RSS values reported by 802.11 wireless cards can be highly inaccurate due to the type of hardware, poor calibration, environmental conditions, location, temperature, multi-path effects, and external interference. Second, it relies on the RSS of decoded packets and hence cannot detect *hidden interferers* which are within interference range, but not within communication range.

To address these limitations, we use PDR measurements to complement the previous RSS technique. During the data subframe of each TDMA frame, the MAC at each node i estimates the PDR of link (i,j) as the fraction of successful MAC transmissions. At the end of the TDMA frame, the

MAC reports these measurements to the mesh controller (MC). This information is then used by the MC to update the conflict graph at each TDMA frame, as we show next.

5.2 Conflict Graph Update

The MC uses a conflict graph to represent interference in the network. A vertex v_{ij} in the conflict graph corresponds to the link (i, j) in the network graph. An edge between vertices v_{ij} and v_{kl} denotes interference between links (i, j) and (k, l) in either the DATA or ACK directions. The vertex independent sets in the conflict graph correspond to the link transmission sets.

The conflict graph update mechanism is executed at each TDMA frame, after the MC receives the RSS and PDR measurements. The conflict graph construction occurs in two stages. The first stage uses the RSS measurements and the second stage refines it further with the PDR measurements.

First stage: Recall that S_{ij} is the RSS of link (i, j) measured at node j once per frame. If S_{ij} is not measured, the transmission from i to j is estimated outside of communication and interference range. First, the MC creates a vertex v_{ij} in the conflict graph if $S_{ij} \geq \gamma_j^R$, the RX sensitivity threshold of receiver j at PHY rate R. Then, the MC adds an edge between each pair of vertices v_{ij} and v_{kl} in the conflict graph if either they share a common node or if the SIR of DATA or ACK directions is less than the SIR receiver threshold δ_j^R at PHY rate R.

Second stage: For each link (i, j) selected by the first stage, the MC checks its reported PDR value P_{ij}. If $P_{ij} \geq 90\%$, the link remains in the conflict graph. Otherwise, the MC finds the link transmission set I_{ij} where this link failed the most, by inspecting the schedule of the previous TDMA frame. Among the links in I_{ij}, some were outside the interference range and some were hidden interferers. The MC identifies the hidden interferers using the connectivity graph. Using this graph, one-hop neighbors are defined as the nodes from which a beacon is received; two-hop neighbors are the one-hop neighbors of these neighbors, and so on. To reduce the search space, we assume that the hidden interferers of link (i, j) are those transmitters in I_{ij} which are two-hop neighbors of either i or j. For each such node k, the algorithm adds an edge between vertex v_{ij} and vertex v_{kl} in the conflict graph. If link (i, j) fails again in the same set I_{ij} during the next TDMA frame, the three-hop neighbors of i and j can be considered, until the hidden interferers are detected.

6. TESTBED EVALUATION

In this section, we evaluate XPRESS in an indoor testbed. We first describe our testbed and experimental methodology (Section 6.1), followed by results in single-hop (Section 6.2) and multi-hop scenarios (Section 6.3). Finally, we validate the estimations in our speculative scheduling (Section 6.4).

6.1 Experimental Methodology

Figure 8 shows our 8-node testbed deployed in one floor of our premises. This is a typical office environment with cubicles and glass meeting booths. Each node is a PC with 1.6 GHz CPU, 512MB RAM, and 80GB HD for storing measurement data. Each node runs the XPRESS software in the Linux kernel 2.6.28 and has a Technicolor WiFi card with a proprietary firmware that runs our multi-hop TDMA MAC protocol detailed in [9]. The WiFi cards are set to operate at 5 GHz in the 802.11a frequency band. The MC node is a dedicated Linux PC with a quad-core 2.7 GHz AMD Opteron® computer, with 16 GB RAM. The execution time of our MWIS implementation on the MC did not exceed 100 μs, which means that we can support TDMA slots of size 100 μs and above.

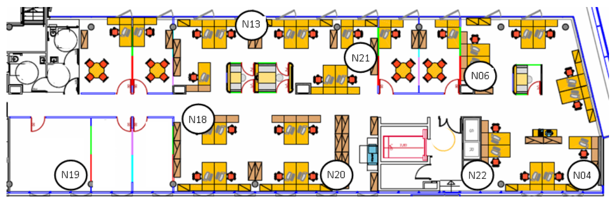


Figure 8: Our wireless indoor testbed (40×8 m^2).

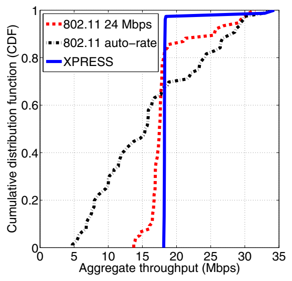
In our testbed, the TDMA control subframe consists of 8 slots of 625 μs each, resulting in a total of 5 ms. In these slots, each node broadcasts its clock synchronization beacon at 6 Mbps PHY rate to other mesh nodes in the testbed, based on the protocol in [9]. The TDMA data subframe consists of 152 slots of 625 μs each, for a total of 95 ms. This results in a TDMA frame duration of 100 ms and a control overhead of 5%.

In our current implementation, the uplink and downlink control channels are implemented using a reliable Ethernet connection instead of wireless transmissions in the TDMA control subframe. In our evaluation, however, we take the actual control overhead into account by making each slot in the control subframe larger than the duration required by the synchronization beacon. According to our analysis in Section 7, the 5-ms control subframe provides enough time to transmit all uplink and downlink control information at 6 Mbps in our 8-node testbed.

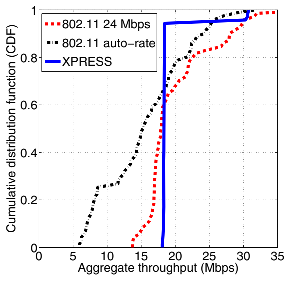
Due to our dense deployment, we take the following steps to enable multi-hop routing and spatial reuse. First, we use the minimum transmission power of our WiFi platform (12 dBm). Second, we experimentally determine 24 Mbps as the highest PHY rate that can create multi-hop topologies without severely compromising link reliability. These power and PHY rate settings result in a total of 20 single links with a PDR higher than 90%, and in 20 link pairs with a PDR higher than 90% for each link when transmitting simultaneously. This provides enough spatial reuse for our testbed size and density.

In our experiments, we compare XPRESS to 802.11 DCF. For XPRESS, we fix the PHY rate for the data subframe to 24 Mbps. At this rate, the SIR threshold is defined as 17 dB based on the profile of RSS versus PDR collected for each link. The profile was constructed by offline measurements of links transmitting alone and in pairs. The 17 dB was the highest threshold that enabled a PDR greater than 90% in most cases. We set the upper limit of XPRESS FlowQs to 400 packets. The congestion control parameter K was experimentally determined as 100, a value that allowed a single link to achieve its full capacity for our logarithmic utility function.

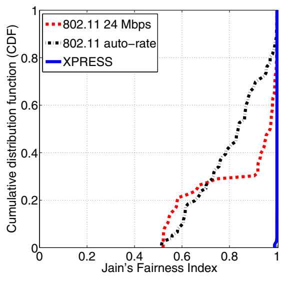
For 802.11 DCF we use both a fixed 24 Mbps PHY rate and the automatic PHY rate adaptation scheme of our WiFi card (noted as *auto-rate* hereafter). We also disable the RTS/CTS exchange because it creates a high overhead for 802.11 DCF at the relatively high rate of 24 Mbps.



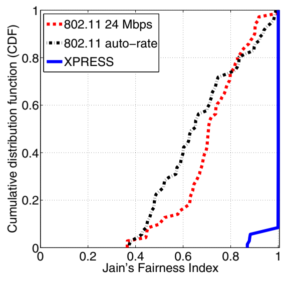
(a) 2 APs, 1 client each.



(b) 3 APs, 1 client each.



(c) 2 APs, 1 client each.



(d) 3 APs, 1 client each.

Figure 7: The throughput and fairness in centralized WLAN scenarios.

In order to maintain repeatability across different testruns, we select a 5-GHz channel free of external interference and set the MAC retransmission limit to 7 for both XPRESS and 802.11. We use *Iperf* [7] to generate UDP traffic with 1470-byte payload packets and measure throughput as the goodput received at the flow destination.

6.2 Single-Hop WLAN Performance

In WLAN scenarios, XPRESS can operate as a centralized WLAN architecture, such as a wireless home network with a few APs controlled by the home gateway or an enterprise WLAN with lightweight APs and a central controller.

We consider several WLAN scenarios and present results for 2 APs and 3 APs, each with 1 client. Their results are representative for other scenarios that we tested. For each scenario, we considered all possible combinations of nodes in our testbed.

Figures 7(a) and 7(b) depict the aggregate throughput for both XPRESS and 802.11. The figures show that XPRESS yields roughly 18 Mbps for most configurations in both scenarios. This is the maximum UDP throughput of a single link at 24 Mbps PHY rate, hence, XPRESS scheduled links in sequence. Figures 7(c) and 7(d) depict the cumulative distribution function (CDF) of the Jain's Fairness Index (JFI) and show that the JFI of XPRESS is always close to unity, demonstrating almost perfect fairness.

We see that the aggregate XPRESS throughput is spread below and above the 802.11 throughput. When XPRESS performs sequential scheduling, 802.11 may achieve a higher throughput in two cases: when it performs successful parallel scheduling and/or when 802.11 auto-rate increases the PHY rate above 24 Mbps. For example, in Figure 7(a) parallel scheduling in 24-Mbps 802.11 achieves higher throughput than XPRESS for 17% of the configurations; the higher PHY rates discovered by 802.11 auto-rate increased this fraction to 35%. However, for the majority of the configurations (83% and 65% respectively), XPRESS achieves much higher throughput, mainly due to successfully coping with hidden terminals and collisions. Figure7(b) shows the same trend and we observed the same in other scenarios we tested.

XPRESS schedules links in parallel in a few configurations. In these cases, XPRESS doubles the throughput, while the best 802.11 results are slightly lower due to 802.11 fairness issues, shown in Figures 7(c) and 7(d). However, XPRESS achieves perfect fairness in these scenarios.

In summary, XPRESS can provide stable throughput and almost perfect fairness in small-scale WLANs, while still achieving higher throughput than 802.11. On the other hand, 802.11 achieves variable throughput and, when 802.11 achieves a higher throughput than XPRESS, it may result in unfairness.

6.3 Multi-Hop Mesh Performance

We now evaluate XPRESS in multi-hop mesh scenarios, consisting of line, gateway-centric, and multi-path scenarios.

Line scenario: We consider several single-path/single-flow configurations, ranging from one to six hops. For each configuration, we restrict the scheduler to use only the links in the path and perform repeated testruns with a 10-Mbps UDP traffic load. Figure 9 depicts the average throughput of XPRESS and 802.11 versus the number of hops. The throughput of XPRESS decreases rapidly with the number of hops until 4 hops, where spatial reuse on the path starts being exploited. On the other hand, 802.11 continues to decrease rapidly even after 4 hops. The relative improvement of XPRESS over 802.11 increases with the path length. For instance, for an offered load of 10 Mbps, 802.11 achieves 3.3 Mbps at 4 hops, 2.2 Mbps at 5 hops, and 1.7 Mbps at 6 hops. On the other hand, XPRESS is able to transmit at 4.3 Mbps at 4 hops (30% gain), 4.2 Mbps at 5 hops (91% gain), and 3.5 Mbps at 6 hops (106% gain). This gain is due to the hidden/exposed terminals in 802.11 and the spatial reuse in XPRESS, which starts at 5 hops, where the aggregate throughput is $5 \times 4.2 = 21$ Mbps, as opposed to $4 \times 4.3 = 17.2$ Mbps at 4 hops. These results demonstrate the benefits of XPRESS backpressure scheduling over 802.11 DCF in achieving the full path capacity by efficient coordination of transmissions.

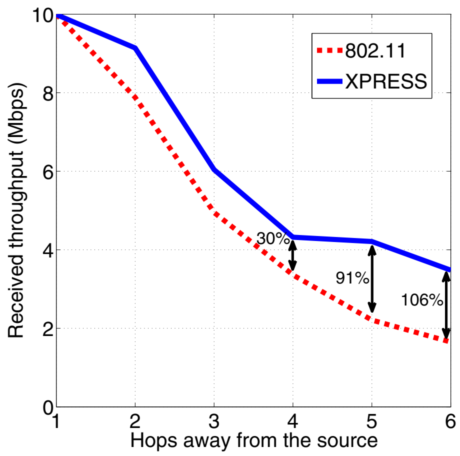
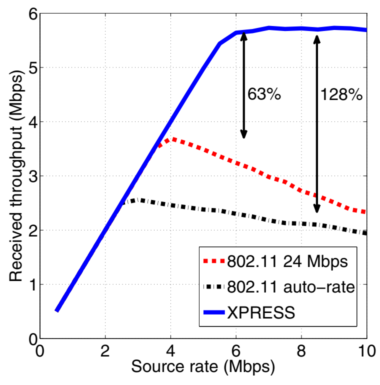
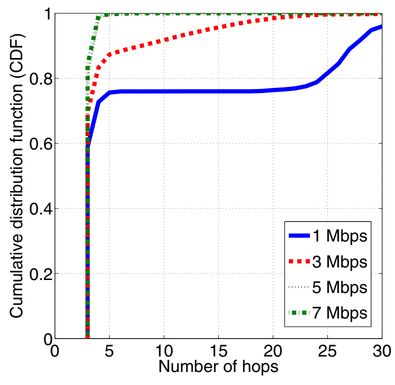


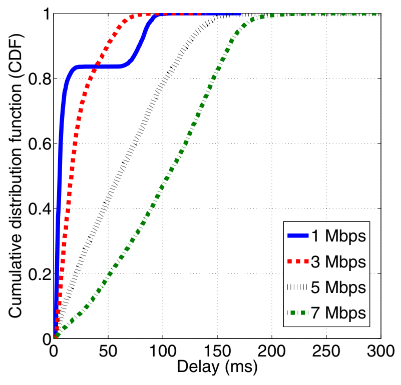
Figure 9: The throughput of a line topology for both 802.11 and XPRESS for an offered load of 10 Mbps.



(a) Throughput at the receiver.



(b) Number of hops per packet.



(c) Network delay.

Figure 10: The throughput, number of hops, and delay for the multi-path experiment.

Gateway-centric scenario: We consider a typical mesh scenario of a 2-hop gateway-centric topology with 2 flows, $A \rightarrow B \rightarrow C$ and $B \rightarrow C$, originating from two MAPs, A and B, toward a gateway C over a single path. Both flows have the same UDP source rate. Figure 11 presents the per-flow throughput versus the offered load of each flow.

From the figure, we can see that both flows can simultaneously sustain up to 5 Mbps offered load each, for both XPRESS and 802.11. Above this, the aggregate throughput of 802.11 decreases and unfairness occurs at the expense of the 2-hop flow. On the other hand, XPRESS achieves consistently higher per-flow and aggregate throughput than 802.11. We can see that XPRESS serves both flows fairly until 6 Mbps. This is roughly the capacity of the network, where the 2-hop flow consumes 6 + 6 = 12 Mbps and the 1-hop flow consumes 6 Mbps (i.e., a total of 18 Mbps). As the source rate increases, we see that the medium time is still equally shared between the two flows. However, the longer flow needs two hops to reach the gateway, which reduces its throughput by half. On the other hand, the lack of coordination in 802.11 results in wireless losses due to severe contention, resulting in high throughput loss. At the highest source rate, the aggregate 802.11 throughput is only 7.9 Mbps, while XPRESS achieves 14.0 Mbps (77% gain).

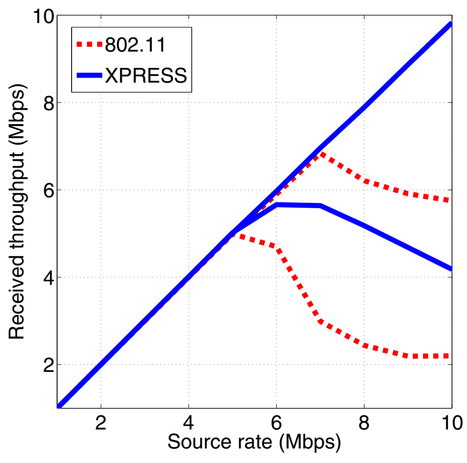


Figure 11: The throughput of two flows $A \rightarrow B \rightarrow C$ and $B \rightarrow C$ in a GW-centric 2-hop topology.

Multi-path scenario: We now investigate the ability of XPRESS to exploit multi-path capabilities in our testbed. In these experiments, packets may travel different paths between the same source and destination, depending on the per-slot instantaneous differential queue backlogs. We investigate two different aspects: throughput and end-to-end delay. We set up a UDP flow between nodes 4 and 19, the farthest nodes in our testbed, and allow the XPRESS scheduler to use all possible links in the testbed. We compare XPRESS with 802.11 at fixed 24 Mbps PHY rate and auto-rate using the best path computed based on the ETX routing metric, through nodes 4, 22, 21, 18, 19 (cf. Figure 8).

Figure 10(a) depicts the received throughput at node 19 versus the input source rate at node 4. The throughput of XPRESS increases linearly with the offered load until 5.5 Mbps, after which it remains stable at the maximum of 5.7 Mbps. On the other hand, 802.11 reaches only 3.5 Mbps (63% gain for XPRESS) with a fixed rate of 24 Mbps and 2.5 Mbps (128% gain for XPRESS) with auto-rate, after which throughput declines. The decline in 802.11 at high input rate occurs because of hidden terminal collisions along the 4-hop path, which trigger packet retransmissions and reduce the end-to-end throughput. XPRESS does not suffer from hidden terminals and is able to sustain the maximum throughput. From the figure, we can also see that 802.11 auto-rate receives less throughput than 802.11 with 24 Mbps under high load. This occurs because the collisions often cause auto-rate to fall back to low PHY rates.

We now investigate the delay properties of XPRESS. In Figure 10(b), we see the cumulative distribution function (CDF) of the path hop count taken by each packet, and in Figure 10(c) we see the delay measured after the rate controller at the source. Figure 10(b) shows that, under high loads, almost all packets follow 3-hop or 4-hop paths. As the load decreases, however, an increasing fraction of packets follows longer paths. At the lowest load of 1 Mbps, a fraction of 20% of the packets followed a path between 25 and 30 hops. The reason is that queues are small and the differential backlogs are not effective in path differentiation; this is an inherent property of backpressure scheduling. However, as shown in Figure 10(c), the delay of the slowest packets under 1 Mbps load does not exceed 100 ms. Thus, despite the long paths taken, once these packets exit the congestion controller and enter the backpressure scheduler, they are delivered to the destination in less than a TDMA frame duration. Delays increase after 5 Mbps load, which is close to the capacity limit of 5.7 Mbps shown in Figure 10(a). However, for all offered loads beyond the capacity limit, the additional delay is limited to about 50 ms. In this case, the FlowQs are always full and the scheduler delay is dominated by the maximum FlowQ buffer sizes. Also, the delays are finite, which indicates that the congestion controller feeds the backpressure scheduler with rates within the network capacity region.

6.4 Validating Queue Estimations

We now evaluate the queue estimation accuracy of the speculative technique in our scheduler (cf. Section 3.4). At each slot in the last 60-second multi-path experiment, we collect the estimated FlowQ backlog $\hat{Q}$ and the actual FlowQ backlog Q of each node, obtaining more than 700,000 samples. Figure 12 shows the cumulative distribution function (CDF) of the error $\hat{Q} - Q$ in the queue backlog estimates for all samples during the experiment. We see that the maximum error is only 5 packets, which is very small compared to the tens or hundreds of packets backlogged at the FlowQs at each node. This validates that XPRESS estimates queue backlogs correctly. Moreover, the small error supports the claim that the the throughput achieved by XPRESS in Figure 10(a) is indeed very close to the network capacity, in addition to being significantly higher than 802.11.

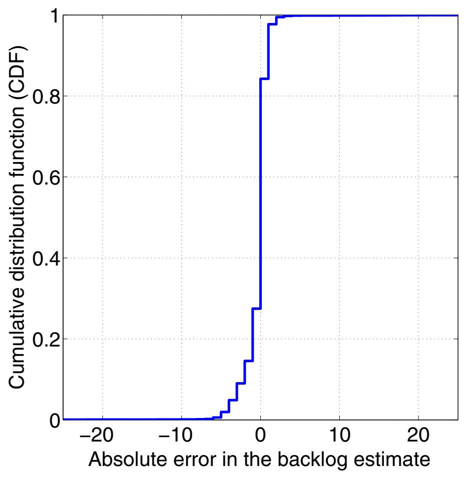
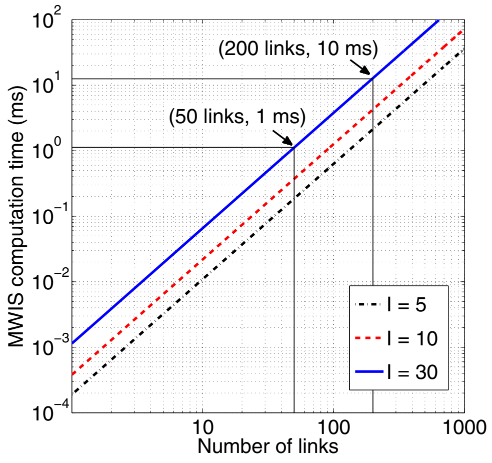


Figure 12: The cumulative distribution function (CDF) of the absolute error in the backlog estimate $\hat{Q} - Q$ for each node at each slot. The estimates are close to the actual queue lengths, indicating that XPRESS achieves the network capacity.

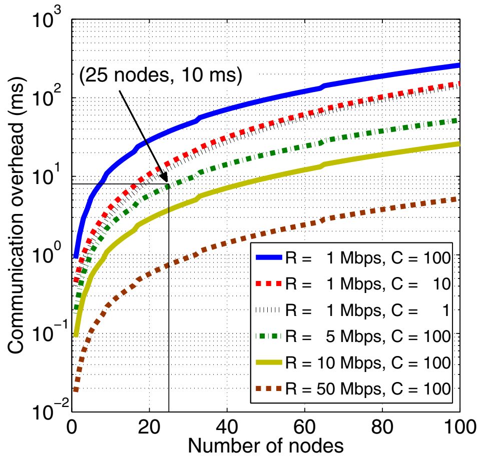
7. OVERHEAD ANALYSIS

We now analyze the processing and communication overhead in XPRESS. Our goal is to understand how it scales for larger, yet practical, networks (e.g., 20, 50, 100 nodes) and the corresponding design trade-offs.

Computation overhead: In theory, the maximum weight independent set (MWIS) problem is NP-hard. However, Wang [26] experimentally shows that for practical MWIS problems in wireless multi-hop networks, the computation



(a) The mean MWIS computation time as a function of the number of links in the network. The implementation was on CPLEX running on top of an Intel 2.83GHz E5440 CPU with 16 GB RAM.



(b) The per-frame communication overhead as a function of the number of MAPs, for values of C and R. We assume $B = 256$, $D = 10$, $Q = 64$, $N_s = 50$, and $N_p = 10$.

Figure 13: The (a) computation overhead at the MC and (b) the communication overhead of XPRESS.

time increases polynomially with the number of links and linearly with the average number of interferers of each link.

Figure 13(a) is derived from the data in [26] and shows the mean MWIS computation time versus the number of links, for average densities I of 5, 10, and 30 interferers per link. With this figure, one can make design decisions about the frequency of the MWIS computation and the degree of multi-path routing. For example, consider a network with 25 nodes and 1-ms TDMA slots. Figure 13(a) shows that the MWIS computation in a very dense ($I = 30$) network of 50 links takes 1 ms, hence it can run for every time slot. At the same density, the MWIS computation for 200 links takes 10 ms and hence can run once every 10 slots. In this case, the intermediate 10 slots would either reuse the last computed MWIS schedule or resort to heuristics to compute potentially sub-optimal schedules. Another alternative is for the MC to prune a few links, which reduces the MWIS computation time and allows per-slot computations at the cost of less available paths.

Communication overhead: The communication overhead is the time to transmit the control information on the uplink and downlink control channels (cf. Figure 5). Assume that these channels can deliver this information at R bps. The downlink channel disseminates the synchronization beacon and C schedules for all N nodes. Each schedule specifies the selected set of links and the corresponding flows allowed to transmit. In the worst case, all nodes are selected for transmission in the same schedule. Thus, each schedule can be encoded with at most $N(\log_2 F + \log_2 D)$ bits, where F is the maximum number of flows and D is the maximum node degree in the network. If B bits are used for the synchronization beacon, an upper bound on the downlink time is

$$T_D = \frac{CN(\log_2 F + \log_2 D) + B}{R}. \quad (5)$$

The uplink overhead consists of N control packets sent by the nodes to the MC. Each packet contains the queue backlogs of all flows at the node, as well as the ID, the RSS, and the PDR of each neighbor. The uplink time T_U is thus

$$T_U = \frac{N}{R}\left[F \log_2 Q + D(\log_2 N + \log_2 N_s + \log_2 N_p)\right], \quad (6)$$

where Q is the maximum FlowQ size, N_s is the number of RSS levels measured by the wireless cards, and N_p is the number of PDR levels.

If we assume that there are only flows between the nodes and the gateway (i.e., $F = 2N$), we see from Equations (5) and (6) that, for large network sizes, the uplink overhead $O(NF)$ is higher than the downlink overhead $O(N \log_2 F)$. However, for practical network sizes, they may be comparable depending on C.

Figure 13(b) plots the total control overhead time $T_U + T_D$ as a function of N for values of R and C, assuming $F = 2N$. With this figure, one can make design decisions between the number of schedules C and the rate R. For instance, when $R = 1$ Mbps, we see that increasing C from 10 to 100 only doubles the total overhead, i.e., same effect as halving R. This holds for any R and it occurs because, when $C = 100$, the uplink and downlink overhead become comparable. For the 25-node XPRESS network previously discussed, assume a requirement of 1-ms TDMA slots, with a control subframe of 10 ms and a data subframe of 100 ms (i.e., 10% overhead). Figure 13(b) shows that this requirement can be satisfied with $R \geq 5$ Mbps and $C = 100$ schedules.

8. RELATED WORK

Backpressure architectures: The backpressure algorithm was introduced by Tassiulas and Ephremides [24] and, since then, a significant effort has been dedicated to distributed approximations which guarantee a fraction of the capacity region [12]. These algorithms maintain the slotted TDMA MAC protocol assumption of the original algorithm, but have not yet been translated to real implementations. On the other hand, recent work applies backpressure ideas to build practical systems on top of existing MAC protocols, such as 802.11. Akyol *et al.* [2] modify the 802.11 contention window to prioritize links with a higher differential backlog. Radunovic *et al.* [20] enhance the performance of multi-path TCP transfers with a simplified backpressure scheme on top of 802.11. In a similar fashion, Aziz *et al.* [3], Ryu *et al.* [22], and Warrier *et al.* [27] approximate backpressure scheduling using prioritization in 802.11. Moeller *et al.* [16] build a backpressure routing architecture over 802.15.4 to enhance data collection in wireless sensor networks. Most of these works [2, 3, 20, 27] assume a separate routing protocol.

These approaches were important steps to show practical improvements of backpressure-inspired protocols over regular CSMA/CA. However, it is not clear how close they are to the throughput optimality of XPRESS, since they are constrained by the CSMA/CA MAC interference estimation limitations, such as hidden and exposed terminals.

Centralized 802.11 architectures: Vendors, like Cisco, Aruba, and Meru Networks, offer centralized architectures for enterprise 802.11 wireless LANs. In these architectures, a central controller coordinates downlink transmissions from the APs to the clients. With a similiar goal, Liu *et al.* [13] and Srivastava *et al.* [23] propose centralized architectures to schedule AP transmissions. XPRESS shares the centralized philosophy of these architectures, but also has fundamental differences. For instance, these architectures only schedule single-hop transmissions from the AP to the clients and use heuristics for scheduling. On the other hand, XPRESS controls the entire multi-hop wireless backbone and implements a throughput-optimal scheduler.

Interference estimation: Existing techniques for interference estimation can be broadly classified as passive or active. Passive approaches require monitors deployed throughout the wireless network to collect traffic traces, which are later analyzed offline [5, 14]. Active approaches, on the other hand, use the available infrastructure to inject test packets into the network and measure interference. Padhye *et al.* [17] measure interference by comparing the throughput of two links, when transmitting in isolation and in parallel. Altough accurate, this technique requires extensive downtime to test all link pairs. Ahmed *et al.* [1] propose an online approach where APs periodically silence their clients and run a quick interference tests. This technique, however, is limited to the downlink of single-hop wireless LANs. For multi-hop networks, Vutukuru *et al.* [25] propose a passive technique to learn about transmission conflicts. This approach exploits several exposed terminals unused by 802.11, but it does not address hidden interferers. Finally, several works measure received signal strength (RSS) to predict packet delivery ratios (PDR) [13, 18, 21]. In addition to RSS measurements, the XPRESS interference estimation design measures PDR during each TDMA frame to detect hidden interferers and refine the network conflict graph.

9. DISCUSSION

While XPRESS has shown significant improvements over 802.11, there are a few issues that must be dealt with for efficient operation. In this section, we describe these issues and present new research directions inspired by XPRESS.

TCP support: XPRESS operates at the IP layer and therefore any transport protocol, including TCP, can run on top of XPRESS. Efficient TCP operation, however, requires handling out-of-order packets, due to the backpressure multi-path routing, and supporting variable TCP packet sizes over the XPRESS TDMA MAC. Out-of-order packets have been addressed with delayed packet delivery in [20] and variable packet sizes could also be addressed with extra implementation effort. Integrating TCP with an optimal backpressure scheduler and understanding their interactions, however, is a challenge for future research.

Rate adaptation: The XPRESS design applies to both single-rate and multi-rate systems. For instance, the binary conflict graph representation holds for multiple rates, and the RSS-based interference measurement complexity $O(N)$ is independent of the rate. However, the main challenge in multi-rate systems is that the MWIS computational complexity increases with the number of rates. We believe that efficient rate adaptation in a centralized multi-hop network like XPRESS is a challenging topic for future investigation.

10. CONCLUSIONS

We presented the design and implementation of XPRESS, a backpressure architecture for wireless multi-hop networks. Our design leverages a centralized controller for obtaining throughput-optimal scheduling. In contrast to previous work, we integrated backpressure scheduling with a TDMA MAC protocol to allow precise timing in transmissions. Moreover, we introduced a novel interference estimation technique and an efficient speculative backpressure scheduler. Our results in an indoor testbed confirm that XPRESS achieves the full network capacity. Already in seemingly simple network configurations with limited spatial reuse, XPRESS shows up to 128% gains over 802.11. In our future work, apart from TCP and multi-rate support, we intend to evaluate XPRESS in larger networks and conduct a comparison with CSMA/CA backpressure implementations. We believe our work opens up interesting avenues in wireless network system design, showing that optimal centralized routing and scheduling are feasible for small- to medium-sized wireless multi-hop networks. For larger networks, we believe that the design primitives of the XPRESS cross-layer protocol stack can lead to new distributed wireless architectures beyond 802.11.

11. ACKNOWLEDGMENTS

We would like to thank our shepherd, Bhaskaran Raman, and the anonymous reviewers for their valuable feedback, which helped to improve our paper. This work was supported by the OPNEX project of the European Community Seventh Framework Programme (FP7-ICT-224218).

12. REFERENCES

[1] N. Ahmed, U. Ismail, S. Keshav, and K. Papagiannaki. Online Estimation of RF Interference. In *Proc. ACM CoNEXT*, Dec. 2008.

[2] U. Akyol, M. Andrews, P. Gupta, J. Hobby, I. Saniee, and A. Stolyar. Joint Scheduling and Congestion Control in Mobile Ad-Hoc Networks. In *Proc. IEEE INFOCOM*, Apr. 2008.

[3] A. Aziz, D. Starobinski, P. Thiran, and A. ElFawal. EZ-Flow: Removing Turbulence in IEEE 802.11 Wireless Mesh Networks without Message Passing. In *Proc. ACM CoNEXT*, Dec. 2009.

[4] L. Chen, S. Low, M. Chiang, and J. Doyle. Cross-Layer Congestion Control, Routing and Scheduling Design in Ad Hoc Wireless Networks. In *Proc. IEEE INFOCOM*, Apr. 2006.

[5] Y.-C. Cheng, J. Bellardo, P. Benkö, A. Snoeren, G. Voelker, and S. Savage. Jigsaw: Solving the Puzzle of Enterprise 802.11 Analysis. In *Proc. ACM SIGCOMM*, 2006.

[6] Atilla Eryilmaz and R. Srikant. Joint Congestion Control, Routing, and MAC for Stability and Fairness in Wireless Networks. *IEEE Journal on Sel. Areas in Comm.*, 24(8):1514–1524, Aug. 2006.

[7] iperf. http://dast.nlanr.net/projects/iperf.

[8] F. Kelly, A. Maulloo, and D. Tan. Rate Control in Communication Networks: Shadow Prices, Proportional Fairness and Stability. *Journal of the Operational Research Society*, 49:237–252, 1998.

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

[10] J. Lee, J. Ryu, S. Lee, and T. Kwon. Improved Modeling of IEEE 802.11a PHY through Fine-grained Measurements. *Computer Networks*, 54(4), Mar. 2009.

[11] X. Lin and N. B. Shroff. Joint Rate Control and Scheduling in Multihop Wireless Networks. In *CDC*, Dec. 2004.

[12] X. Lin and N. B. Shroff. The Impact of Imperfect Scheduling on Cross-Layer Congestion Control in Wireless Networks . *IEEE Transactions on Networking*, 14(2):302–315, Apr. 2006.

[13] X. Liu, A. Sheth, M. Kaminsky, K. Papagiannaki, S. Seshan, and P. Steenkiste. DIRC: Increasing Indoor Wireless Capacity Using Directional Antennas. In *Proc. ACM SIGCOMM*, Aug. 2009.

[14] R. Mahajan, M. Rodrig, D. Wetherall, and J. Zahorjan. Analyzing the Mac-Level Behavior of Wireless Networks in the Wild. In *Proc. ACM SIGCOMM*, Aug. 2006.

[15] K. Makino and T. Uno. New algorithms for enumerating all maximal cliques. In *In Proc. 9th Scandinavian Workshop on Algorithm Theory*, Humlebaek, Denmark, Jul. 2004.

[16] S. Moeller, A. Sridharan, B. Krishnamachari, and O. Gnawali. Routing Without Routes: The Backpressure Collection Protocol. In *Proc. IEEE/ACM IPSN*, Apr. 2010.

[17] J. Padhye, S. Agarwal, V. Padmanabhan, L. Qiu, A. Rao, and B. Zill. Estimation of Link Interference in Static Multi-hop Wireless Networks. In *Proc. ACM Internet Measurement Conference (IMC)*, Oct. 2005.

[18] L. Qiu, Y. Zhang, F. Wang, M. Han, and R. Mahajan. A General Model of Wireless Interference. In *Proc. ACM MobiCom*, Oct. 2007.

[19] B. Radunovic and Jean-Yves Le Boudec. Rate Performance Objectives of Multihop Wireless Networks. *IEEE Transactions on Mobile Computing*, 3(4):334–349, 2004.

[20] B. Radunovic, C. Gkantsidis, D. Gunawardena, and P. Key. Horizon: Balancing TCP over Multiple Paths in Wireless Mesh Network. In *Proc. ACM MobiCom*, Sep. 2008.

[21] C. Reis, R. Mahajan, M. Rodrig, D. Wetherall, , and J. Zahorjan. Measurement-Based Models of Delivery and Interference. In *Proc. ACM SIGCOMM*, Sep. 2006.

[22] J. Ryu, V. Bhargava, N. Paine, and S. Shakkottai. Back-Pressure Routing and Rate Control for ICNs. In *Proc. ACM MobiCom*, Sep. 2010.

[23] V. Shrivastava, N. Ahmed, S. Rayanchu, S. Banerjee, S. Keshav, K. Papagiannaki, and A. Mishra. CENTAUR: Realizing the Full Potential of Centralized WLANs through a Hybrid Data Path. In *Proc. ACM MobiCom*, Sep. 2009.

[24] L. Tassiulas and A. Ephremides. Stability Properties of Constrained Queuing Systems and Scheduling Policies for Maximum Throughput in Multihop Radio Networks. *IEEE Transactions on Automatic Control*, 37(12):1936–1948, Dec. 1992.

[25] M. Vutukuru, K. Jamieson, and H. Balakrishnan. Harnessing Exposed Terminals in Wireless Networks. In *Proc. NSDI*, Apr. 2008.

[26] P. Wang. Throughput Optimization of Urban Wireless Mesh Network. In *Ph.D. Thesis*, Department of Electrical and Computer Engineering, University of Delaware, 2009.

[27] A. Warrier, S. Janakiraman, S. Ha, and I. Rhee. DiffQ: Practical Differential Backlog Congestion Control for Wireless Networks. In *Proc. IEEE INFOCOM*, Apr. 2009.