Horizon: Balancing TCP over multiple paths in wireless mesh network

Božidar Radunović, Christos Gkantsidis, Dinan Gunawardena, Peter Key

ABSTRACT

There has been extensive work on network architectures that support multi-path routing to improve performance in wireless mesh networks. However, previous work uses ad-hoc design principles that cannot guarantee any network-wide performance objectives such as conjointly maximizing resource utilization and improving fairness. In parallel, numerous theoretical results have addressed the issue of optimizing a combined metric of network utilization and fairness using techniques based on back-pressure scheduling, routing and flow control. However, the proposed theoretical algorithms are extremely difficult to implement in practice, especially in the presence of 802.11 MAC and TCP.

We propose *Horizon* , [1] a novel system design for multipath forwarding in wireless meshes, based on the theoretical results on back-pressure. Our design works with an unmodified TCP stack and on the top of the existing 802.11 MAC. We modified the back-pressure approach to obtain a simple 802.11-compatible packet-forwarding heuristic and a novel, light-weight path estimator, while maintaining global optimality properties. We propose a delayed reordering algorithm that eliminates TCP timeouts while keeping TCP packet reordering at a minimum. We have evaluated our implementation on a 22-node testbed. We have shown that *Horizon* effectively utilizes available resources (disjoint paths). In contrast to previous work our design can not only avoid bottlenecks but also optimally load-balances traffic across them when needed, improving fairness among competing flows. To our knowledge, *Horizon* is the first practical system based on back-pressure.

1. INTRODUCTION

Wireless networks are very easy to deploy, but difficult to make work effectively. End-to-end performance can be very poor due to variable link quality, caused by interference and changes in the environment. This is especially true for wireless mesh networks where packets traverse several consecutive links. The main design challenge is to schedule links and route packets in order to efficiently use network resources while guaranteeing fairness among users. Wireless mesh networks typically provide several paths from a source to a destination, and by using such paths efficiently we can aggregate the available resources. This has the potential not only to increase multiplicatively the achieved end-to-end rate, but also to provide robustness against performance fluctuations of any single link in the system.

There is a large body of recent theoretical work that explores back-pressure scheduling [20] in context of utility maximization [18]. The *back-log* is represented by the queue sizes at nodes, and the main idea of back-pressure scheduling is to give priority to links and paths that have higher back-pressure, defined as the differential back-log at consecutive nodes. If we assign a 'utility' to each flow which is a function of the flow's rate, then utility maximization seeks to design network protocols that will maximize the aggregate utility.

Several algorithms based on back-pressure have been proposed,which have been shown to solve the network-wide utility maximization problem (c.f. [5, 6, 8, 12, 14]). These algorithms comprise a routing algorithm that is able to exploit multiple-paths, a scheduling algorithm that prioritize links' access to the wireless medium, and a flow control algorithm that prevents congestion collapse and guarantee fairness among flows.

There are several benefits of the utility maximization approach. It enables us to allocate rates across paths optimally. Furthermore, prioritized scheduling maximizes the achievable link rates. Finally, utility maximization enables us to address fairness issues. This is an important question, since a mesh network can quickly saturate with a modest number of flows, because adjacent links interfere. Once a network is saturated, one cannot increase total throughput when new flows are added. However, it is possible to ensure fairness. A long flow that traverses several hops will be highly penalized when competing for medium access with a short flow. Protocols based on utility maximization enforce fairness by giving long flows a higher chance of access the wireless medium as well as offering the possibility of increasing their rate by using multiple paths.

However, all proposed algorithms [5, 6, 12, 14] are purely theoretical. Although provably optimal, they are extremely difficult to implement. Firstly, back-pressure scheduling is

[1] Horizon: from Greek – $o\rho\iota\zeta\omega$ – to divide, separate.

NP-hard [8]; all proposed implementations incur prohibitively large signaling overhead and require new MAC protocols. Secondly, any routing and scheduling protocol for wireless mesh will interact with TCP/IP and the rest of the networking stack, and algorithms from [5,6,12,14] do not work well with TCP.

Our focus is 802.11 mesh networks. We describe *Horizon* , a novel network design based on utility maximization approach, back-pressure scheduling and multi-path routing that is interoperable with TCP. We identify critical problems of previous approaches [5, 6, 12, 14] and propose specific solutions. We set out to ensure that (a) our design works with existing MAC protocols and TCP, (b) efficiently uses the wireless resources, and (c) is fair between flows.

There are numerous other proposals for multi-path routing in wireless multi-hop and mesh networks. Our architecture is the first system that strives to maximize some network-wide performance metric using multi-path routing as oppose to each flow maximizing its own performance (e.g. [21]). *Horizon* avoids congested areas when possible but can also optimally balance load across congested area when needed, in contrast to e.g. [17]. In summary, our system

- is implemented as a slim layer between the data link layer and network (i.e. IP) layer,
- can be deployed in existing wireless mesh networks, without changing 802.11 MAC or TCP/IP,
- implements multi-path routing using back-pressure based heuristics for scheduling packets and flows,
- uses a light-weight (i.e. fast and requiring small number of packets queued in the network) path estimator that accurately estimates path quality and calculates back-pressure between nodes,
- employs an algorithm that delivers packets in-order and with a smooth rate at the destination; this is necessary to guarantee that TCP can take advantage of multipath routing, without collapsing its congestion window.

We do not implement a routing protocol. Instead, *Horizon* can be integrated with almost any routing protocol that provides us with multiple-disjoint paths, or providing us with enough neighbor information to enable construction of such paths (e.g. link-state routing protocols).

We have implemented our design, and conducted experiments on an 22 node test-bed. When the number of flows is lower than the number of available resources (disjoint paths) we improve the total throughput of the network up to 100% over single-path TCP. In some cases the performance drops. This is due to our simplified, suboptimal scheduling, based on 802.11 MAC that cannot successfully eliminate contention between multiple paths. We identify and discuss such cases and potential improvements. When the number of flow exceeds the number of disjoint paths, resources are typically already fully utilized. In that case we verify empirically that *Horizon* improves the fairness among competing flows, as predicted by theoretical frameworks. In most instances we see an increase in the system utility and in the rate of the worst flow.

This paper is organized as follows. In Section 2 we give a brief overview of the existing theoretical works on utility maximization in wireless and discuss the open problems. We present our novel path estimation and packet forwarding algorithms in Section 3 and delay-reordering algorithm for interacting with TCP in Section 4. We present detailed system architecture in Section 5 and we evaluate its performance in Section 6. In Section 7 we discuss related work and we conclude in Section 8.

2. UTILITY AND BACK-PRESSURE

There is a large area of research, both in wired and wireless networking, that pose network protocol design as a convex optimization problem (c.f. [5,6,8,12,18]). Here we give a brief outline of the topic, as our design is based on some of these principles. We shall follow the exposition of [12]. After presenting the optimization problem in Sec. 2.1 and a simple example in Sec. 2.2, Sec. 2.3 summarizes the difficulties implementing the standard frameworks in current networks.

2.1 Utility Maximization Problem

Let $\mathcal{N}$ be the set of nodes and $\mathcal{F} \subseteq \mathcal{N}^2$ the set of flows in the network, defined as pairs of ingress and egress nodes. Let x_{ij}^f be the packet rate from flow f on the wireless link from node i to j and let y_f be the rate of fresh packets injected at the source node $s(f)$ of flow f. Let us denote by $\mathcal{U}_f(i), \mathcal{D}_f(i) \subseteq \mathcal{N}$ the sets of upstream and downstream neighbors of node i for flow f. These sets are predefined by an exogenous routing protocol.

Traffic at node i is stable if the total ingress traffic is smaller than the total egress traffic, which we write as

$$\sum_{j \in \mathcal{D}_f(i)} x_{ij}^f - \sum_{j \in \mathcal{U}_f(i)} x_{ji}^f - y_f \, 1_{\{s(f)=i\}} \geq 0 \qquad (1)$$

where $1_{\{s(f)=i\}} = 1$ if $s(f) = i$, or 0 otherwise. We call (1) the flow conservation constraint. We also have

$$x_{ij}^f \geq 0. \qquad (2)$$

Let $\mathcal{R} = \{(r_{ij})_{ij}\}$ be the set of feasible average rates on links (i, j) that can possibly be achieved by any MAC protocol. The rates $\mathcal{R}$ are determined by network topology, channel conditions, interference, etc. In this section we shall assume that we have an ideal MAC that can achieve any rate from this set. Then the MAC layer (scheduling) constraints are

$$\left(\sum_{f \in \mathcal{F}} x_{ij}^f \right)_{ij} \in \mathcal{R}. \qquad (3)$$

Equations (1)-(3) define the set of average flow rates a network can support: the set of flow rates $(y_f)_{f \in \mathcal{F}}$ can be sup-

ported by the network $(\mathcal{N}, \mathcal{F})$ if there exist rates $(x_{ij}^f)_{i,j,f}$ that satisfy constraints (1)-(3).

Let $U_f(y_f)$ be a convex function of a flow's rate that defines the flow's utility. The total network utility is $\sum_f U_f(y_f)$. The goal of utility maximization approach is to find the flow rates $(y_f)_{f\in\mathcal{F}}$ that solve

$$\text{maximize} \sum_f U_f(y_f) \quad \text{subject to } (1)-(3). \qquad (4)$$

The above optimization problem is convex, provided that set $\mathcal{R}$ is convex. As explained in [12] the solution can be obtained via the dual formulation using a gradient descent algorithm. From there we can derive the optimal scheduling, routing and flow control algorithms.

Flow control: The optimal rate of flow f at time t, $y_f^*(t)$ is the solution of

$$y_f^*(t) = \underset{y_f>0}{\operatorname{argmax}} U_f(y_f) - y_f q_{s(f)}(t), \qquad (5)$$

The source of flow f sets the rate of the flow y_f as a function of number of packets in the queue $q_{s(f)}^f$ at the source of flow f, $s(f)$.

Queue Evolution: The evolution of $q_i^f(t)$ is given by

$$q_i^f = \left[q_i^f - \epsilon \left(\sum_{j\in\mathcal{D}_f(i)} x_{ij}^f - \sum_{j\in\mathcal{U}_f(i)} x_{ji}^f - y_f \, 1_{\{s(f)=i\}} \right) \right]^+$$

where $[x]^+ = x$ if $x \geq 0$, and otherwise 0. Variable q_i^f is the Lagrangian corresponding to the constraint (1). As shown in e.g. [6], q_i^f grows in proportion of the excess data arrived at node f that is not forwarded; it has the same evolution as the queue at node i for flow f.

Scheduling and Routing: The optimal routing and scheduling is defined by equation

$$\mathbf{r}^*(t) = \underset{\mathbf{r}\in\mathcal{R}}{\operatorname{argmax}} \sum_{i,j} r_{ij} \max_f (q_i^f(t) - q_j^f(t)), \qquad (6)$$

For example, if $r_{ij}^* = 0$ this represents a routing decision that no packet should be sent from i to j. The difference $q_i^f(t) - q_j^f(t)$ is called *differential backlog*. For every link (i,j) we first select the flow with the maximum differential backlog $f^*(i,j) = \operatorname{argmax}_f (q_i^f(t) - q_j^f(t))$. Then we select the rate vector $\mathbf{r}^*(t)$, maximizing (6), which defines which links should be active and with what rates, and set $x_{ij}^{f^*} = r_{ij}$ and $x_{ij}^f = 0$ otherwise.

The scheduling defined in (6) is called the *back-pressure* or *max-weight scheduling*, originally defined in [20]. It is shown in [5, 6, 12] that in conjunction with (5) it also solves the network-wide optimization problem defined in (4).

2.2 Example

To understand how this scheme works in practice, let us turn to the example from Figure 1. Consider a flow f, going from node 1 to node 6. For purpose of the example, suppose the utility function $U_f(y_f) = -K/y_f$ where $K = 144$. It is easy to verify that the optimal $y_f^* = 6$ and the average

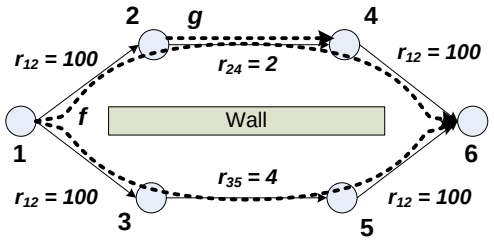

Figure 1: A network with 6 nodes. Flow f goes between nodes 1 and 6, flow g between nodes 2 and 4. Links $r_{24} = 2$ and $r_{35} = 4$ are bottlenecks and links $r_{12} = r_{13} = r_{46} = r_{56} = 100$ are significantly faster.

queue sizes are $q_1^f = q_2^f = q_3^f = 4, q_4^f = q_5^f = 0$ (packets accumulate before bottlenecks 2-4 and 3-5).

Let us suppose that at some time $t = 0$ queue sizes are indeed equal to the average values $q_1^f(0) = q_2^f(0) = q_3^f(0) = 4, q_4^f(0) = q_5^f(0) = 0$, and suppose 9 new packets of flow f arrive at node 1. Because $q_2^f(0) = q_3^f(0)$ node 1 will forward 3 packets to node 2 and 3 packets to node 3. At the next time instant $t = 1$ we will have $q_1^f(1) = q_2^f(1) = q_3^f(1) = 7$. No further packets can be sent because the differential backlogs $q_1^f(1) - q_2^f(1)$ and $q_1^f(1) - q_3^f(1)$ are zero. However, link 3-5 is faster and will get rid of its backlog sooner, and will be ready to receive new packet. This is how on a long term we will have the rates $x_{24}^f = 4$ and $x_{35}^f = 2$.

Next, consider the previous scenario, and assume there is another flow g from node 2 to node 4 that uses only the direct path 2-4. Link 2-4 has lower capacity than link 3-5 hence in this particular example flow f will not route its packet through 1-2-4-6 to maintain fairness. The optimal rate allocations and prices are $y_f^* = 4, y_g^* = 2$ and the average queue sizes are $q_1^f = q_2^f = q_3^f = 9, q_4^f = q_5^f = 0, q_2^g = 36$. Notice that flow f has to queue packets at node 2 only to estimate its quality, although the rate of flow f over link 2-4, $x_{24}^f = 0$. For a more elaborate numerical example, see e.g. [12].

2.3 Difficulties implementing standard framework

The first problem with the back-pressure approach is that the scheduling algorithm (6) is NP hard [8] and extremely difficult to implement in a distributed system. Furthermore, our goal is to implement multi-path routing in an existing 802.11 network without modifying the existing MAC.

The second problem with the back-pressure approach is that it requires a large number of packets to estimate the path quality. In the previous examples, the average total number of queued packets in the network is 12 (in the first example) and 63 (in the second example). Large queues imply large delays. They also limit TCP performance. In the second example, assume that there is only one TCP connection using flow f. Then, we need a TCP window of at least 40kB to be able to correctly estimate the path qualities and achieve the optimal load balancing. This is impossible to achieve since

the window frequently decreases due to wireless losses and reordering (see Section 4) and is above the maximum window size of many OS. In turn, there will be insufficient packets for probing, the source will not be able to correctly estimate the quality of the paths, too many packets will be sent over bad paths, and the system performance will degrade.

3. PATH ESTIMATES AND FORWARDING

In this section, we extend the ideas presented in Section 2 to build (a) a novel light-weight and 802.11-compatible path-quality estimator that requires much few packet in the network to estimate network congestion, and (b) a simple distributed forwarding algorithm.

The main benefits of our approach is the significant reduction in the queue lengths while still maintaining the global properties of the utility-maximization approach. The total number of packets queued in the network become *independent* of the network size and the number of paths, unlike queue sizes in the approach from Section 2 that increase *both* with the network size and the number of paths (see Section 3.5 for a numerical example).

These two proposed algorithms solve a variation of the optimization problem (4). They are simple to implement and, as we shall demonstrate in Section 6, work with the existing TCP and 802.11 implementations and improve performance.

We start by defining a simplified model of the optimization problem (4) and then present our novel path estimation and forwarding algorithms in Section 3.3.

3.1 802.11-compatible Scheduling

Instead of considering the most general form of scheduling given by (6) we shall assume that the underlying 802.11 MAC informs node i whenever it gets an opportunity to transmit. The node should then decide what packet to transmit to which destination. For example, it can monitor the packet queue in the WiFi driver and, when the queue is empty, schedules the next packet. We shall also assume that the node knows which transmission rate 802.11 MAC will use to transmit a packet to the selected destination (these rates are typically decided by MAC layer through some rate control protocol).

We simplify the scheduling constrain (3) in the following way. Suppose the total medium access time T_i for node i, granted by 802.11 MAC, is given and constant. Node i has to decide what and where to transmit during that time. Let R_{ij} be the transmission rate from i to j. Then the fraction of time node i transmits packets from flow f to node j is x_{ij}^f/R_{ij} and we have the following constraint for all nodes i

$$\sum_{f,j\in\mathcal{D}_i(f)} \frac{x_{ij}^f}{R_{ij}} \leq T_i. \tag{7}$$

Obviously, the assumption that T_i is constant does not hold in reality and T_i will depend on the load on i. For example, if node i has no traffic, then it will attempt no transmissions and $T_i = 0$.

3.2 Simplified Model

We next consider the simplified version of (4):

$$\text{maximize} \sum_f U_f(y_f) \quad \text{subject to } (1), (2), (7). \tag{8}$$

where we replaced constraint (3) with (7). Let λ_i^f be the Lagrangian multiplier associated with (1), δ_{ij}^f with (2), and μ_i with (7). From the KKT optimality conditions [2] we derive that $x_{ij}^f > 0$ if and only if

$$R_{ij}(\lambda_i^f - \lambda_j^f) = \mu_i. \tag{9}$$

Eq. (9) determines the pricing policy and the forwarding decisions as follows. Node i will schedule a packet from flow f and next-hop k that maximize the following:

$$\max_{f\in\mathcal{F},k\in\mathcal{D}_i(f)} R_{ik}(\lambda_i^f - \lambda_k^f). \tag{10}$$

The detailed algorithm is given below, in Section 3.3.

In practice, we use a similar dual optimization problem as in Section 2 to design the algorithms for flow and next hop selection, but we optimize the price calculation to avoid excessive queueing; Section 3.5 demonstrates using a simple example some of the inefficiencies of using the queue-based back-pressure algorithm from Section 2.

3.3 Path Estimation and Forwarding

Let P_i^f be the number of packets of flow f queued at i and let C_i^f be the estimated minimal cost of any path for flow f from node i to the destination of path f. Node i calculates C_i^f using the following algorithm:

Path Estimation:

$$j(g) = \operatorname*{argmin}_{j\in\mathcal{D}_i(g)} C_j^g, \tag{11}$$

$$S_i = \max_{g\in\mathcal{F}} P_i^g R_{i,j(g)}, \tag{12}$$

$$C_i^f = S_i + \min_{j:j\in\mathcal{D}_i(f)} C_j^f. \tag{13}$$

Whenever node i gets an opportunity to transmit a packet, it first selects the flow f^* to transmit packet from, and then it selects the node j^* to which it will transmit the packet according to:

Packet Forwarding:

$$f^* = \operatorname*{argmax}_{g\in\mathcal{F},P_i^g>0} P_i^g \tag{14}$$

$$j^* = \operatorname*{argmin}_{j\in\mathcal{D}_i(f^*)} C_j^{f^*} \tag{15}$$

The interpretation of the algorithms is as follows: Each neighbor $j \in \mathcal{D}_i(f)$ announce its price C_j^f, for each flow f, to i. Node i will use the flow with the minimal price to decide where to forward packets. In addition, it will use this minimal price to calculate its own price. To this price, it will add S_i (defined in (12)), which is proportional to the

maximum amount of packets queued at node i. Observe that the nodes require information only from their neighbors (and not from the entire network), and hence the algorithms can be implemented in a distributed system.

It is easy to see that S_i corresponds to μ_i and C_i^f to λ_i^f. Also C_i^f corresponds to the queue size q_i^f from Section 2 on all nodes i used by flow f. However, in this case we do not queue C_i^f packets per node and per flow, but only P_i^f, which is significantly smaller, as we will illustrate in the example in Section 3.5.

3.4 Global Properties of Pricing

We next prove some global properties of the proposed algorithm in a simplified setting (fluid model, as in [12]) whose goal is to illustrate that we indeed follow the utility-maximization approach, as outlined below. More elaborate analysis (queue dynamics, convergence) is outside the scope of this paper.

PROPOSITION 1. *Let us consider the fluid model of the network described above and suppose that it is in the steady state with $\lambda_i^f = C_i^f$ and $\mu_i = S_i$. In that case the achieved rates $(y_f)_f, (x_{ij}^f)_{f,i,j}$ maximize (8).*

PROOF. Since the system is in a steady state, it means that constraints (1) and (7) are satisfied with equalities (where rates are positive). Therefore it is easy to verify that $\lambda_i^f = C_i^f$ and $\mu_i = S_i$ satisfy KKT conditions for (8). Since the problem (8) is convex, KKT conditions are sufficient to prove the optimality. □

3.5 Example

To illustrate how this scheme works in practice, we turn again to the example from Figure 1, described in Section 2.2. Again, it is easy to verify that the optimal $y_f^* = 8$ and the average prices are $C_1^f = C_2^f = C_3^f = 4, C_4^f = C_5^f = 0$ and the queue sizes $P_1^f = P_4^f = P_5^f = 0, P_2^f = P_3^f = 4$. The prices in this examples are exactly the same as in the example from Section 2.2 ($C_i^f = q_i^f$). Nevertheless, the number of packets queued in the network is smaller: 8 in this case, as opposed to 12 in the example from Section 2.2. This is because we do not take queue length as the price indicator but we calculate it implicitly.

In the classical back-pressure approach we need to queue packets in each link before the bottleneck and the total number of queued packets *grows with the network size*. In our case the number of queue packets is *independent of the network size* as packets are queued *only* at the bottleneck links.

Next we consider the two flows scenario described in Section 2.2. Again using the calculus from this section we can calculate the average optimal values of rate allocations $y_f^* = 4, y_g^* = 2$, the average prices $C_1^f = C_3^f = 9, C_4^f = C_5^f = 0, C_2^f = C_2^g = 36$ and the average queue sizes $P_3^f = 9, P_2^g = 36, P_1^f = P_2^f = P_4^f = P_5^f = 0$. The prices in this examples as well are exactly the same as in the example from Section 2.2 ($C_i^f = q_i^f$).

Again, the savings are significant: we queue 45 packets, instead of 90 as we did in Section 2.2. This example emphasizes another benefit of our approach over back-pressure: *we do not need to queue packets on a path we do not use*. We do not need to queue packets from flow f on node 2 ($P_2^f = 0$) because the price of node 2, $C_2^f = C_2^g = 36$ is too high for flow f, so it has no need to send packet that way to estimate it further. Thus our approach *does not grow queue sizes as a function of number of paths* we probe.

Finally, our pricing scheme also requires a certain amount of packets in the network to be able to estimate path qualities. This is more important if the flow contains a very small number of TCP flows, which have low TCP window sizes. In this case, the path estimation may be wrong and the load-balancing suboptimal, like in Section 2.3. However, since our pricing scheme requires much fewer packets, this is less likely to happen.

3.6 When Horizon does not work

As already explained, our packet forwarding scheme is only a heuristic intended to solve the global optimization problem constrained to 802.11 scheduling, and there are cases when this heuristic is suboptimal due to self-contention.

There is one obvious case of self-contention where things will go wrong. In the example from Figure 1 consider a single flow from 1 to 6 and suppose link 4-6 uses much lower PHY transmission rate than link 5-6. Ideally, the back-pressure scheduling as defined in (6) will never select link 4-6. However, 802.11 MAC will observe that link 4-6 has a packet to send, and will eventually schedule it, taking the time of the fast link 5-6 and decreasing the flow's rate. Notice that we need to send at least one packet over the path 1-2-4-6 occasionally, as this is the only way to probe the path. However, we will not be aware that we are creating additional contention and decreasing performance. This problem will not exist if we use a single path only, although that single path may traverse the bad link 4-6, should the routing algorithm decide so. One could construct other, more complicated scenarios involving several links, where our approach can deteriorate performance. These scheduling problems are common to all of the multi-path routing protocols in wireless, and are due to suboptimality of the MAC layer.

However, when the number of flows in the network increases, contention among different flows increases substantially, regardless of whether we use single or multi-path routing, and the self-contention effects become less visible. Also, the self-contention effects are elliminated as we increase the number of available wireless channels and network interfaces. We verify experimentally in Section 6 that in both these cases *Horizon* outperforms the single-path TCP.

4. INTERACTION WITH TCP

Interaction of *Horizon* with TCP is twofold. Firstly, TCP needs to react when a network is congested, decrease the window in order to prevent congestion collapse and enforce

fairness. *Horizon* does flow control and sends congestion indicator to TCP whenever it senses a congestion. Secondly, TCP expects to receive packets in order and within some time-frame, to avoid timeouts. It is widely known that this assumption is violated with multi-path routing as each path may incur arbitrary delays. *Horizon* delays packet delivery to minimize reorderings and timeouts.

4.1 Congestion Control and Fairness

One of the goals of TCP is to detect and avoid network congestion. Another goal is to guarantee fairness among flows. TCP achieves both these goals by reacting to packet losses. Each packet loss is treated as a congestion loss, and the congestion window is halved. Faster flows see more congestion losses which guarantee a certain form of fairness. It has been shown in e.g. [18, Chapter 4.2.1] that TCP performs an approximate utility maximization for utility function satisfying $U'(y_f) = \frac{1}{B_f^2}$ where $B_f = y_f \mathrm{RTT}_f$ is the window size and RTT_f is the round-trip time of flow f.

Horizon communicates congestion to sources using the pricing estimates. From KKT conditions for (8), almost identically as (5) in Section 2, we can derive that the optimal window size B_f^* has to satisfy $U'(y_f) = C^f_{s(f)}$, that is $B_f^* = K/\sqrt{C^f_{s(f)}}$ where K is an arbitrary constant. The choice of K does not affect the optimization problem (8) but does affect the system design. Also, we do not know B_f but we can easily estimate it by estimating y_f and RTT_f. We discuss these issues in Section 5.

We signal congestion by sending a congestion indicator to TCP whenever the TCP window size reaches over the optimal size B_f^*. This way we avoid transmitting packets through the network and wasting the wireless resources, only to drop them at the congested queues further down the network.

4.2 Delayed Reordering and Timeouts

It is well known (c.f. [9, 22]) that packet reordering hampers the performance of multi-path TCP as it causes triple duplicate ACKs, potentially keeps the TCP window small, and degrades the performance of the path estimator. However, unlike [22], we are not allowed to change TCP to cope with reordering. Since *Horizon* is positioned below TCP, it can prevent reordering through delayed packet delivery. When possible, it first waits to receive packets in sequence, and then it delivers the whole ordered sequence.

However, if one of the paths has a delay sufficiently larger than others, TCP may timeout waiting for the packets to be delivered in sequence. This problem is especially emphasized in the case of wireless mesh networks, as opposed to infrastructure networks (a client directly connected to several access-points), since each of several hops on the route can introduce an additional delay.

TCP estimates round-trip time using a simple exponentially weighted moving average algorithm [19]. Let $A(p)$ be the estimated RTT at the reception of packet p and let $D(p)$ be the variance. Then the estimations are updated according to the following rules

$$\begin{aligned} A(p+1) &= (1-\alpha)A(p) + \alpha RTT(p+1), \qquad (16)\\ D(p+1) &= (1-\beta)D(p) + \beta\,|A(p) - RTT(p+1)| \end{aligned}$$

where $RTT(p)$ is the round-trip time of packet p. Time out is defined as $RTO(p+1) = A(p) + 4D(p)$ and it means that TCP triggers timeout if the acknowledgement for packet $p+1$ does not arrive $RTO(p+1)$ after it has been sent.

To avoid TCP timeouts, we propose a *delayed reordering* algorithm. *Horizon* keeps receiving packets and delivers them in sequence. At the same time, it keeps its own estimates of one-way delays. Let $t_s(p)$ be the transmission time of packet p from node s according to the clock at s, and let $t_d(p)$ be the reception time of packet p at destination d according to the clock at d. The skewed one-way propagation is $t_r(p) - t_s(p) = T_{sd}(p) + \Delta$, where $T_{sd}(p)$ is the actual one-way propagation and Δ is an unknown clock skew. We estimate the mean skewed one-way propagation delayed $A_{sd}(p)$ and its variance $D_{sd}(p)$ using the same algorithm as in (16).

The delayed-packet reordering works as follows: Suppose packet p_1 is delayed and packets $p_2, p_3, \ldots$ have arrived. If packet p_1 does not arrive by $t_s(p_2) + A_{sd}(p_2) + 4D_{sd}(p_2)$, we then deliver packet p_2. This will cause a duplicate ACK, but it will not decrease the TCP window, and will give more time for the delayed packet to arrive. Next, if packet p_1 does not arrive by $t_s(p_3) + A_{sd}(p_3) + 4D_{sd}(p_3)$, we then deliver packet p_3. The procedure is repeated until packet p_1 arrives, or the buffer is depleted. After three packets (p_2, p_3, p_4) are delivered out of order triple duplicate ACKs will be sent, and TCP sender will retransmit p_1 and halve the congestion window. However, this effect still has less performance implications than the timeout itself.

There are a few potential problems with the delayed packet reordering. Firstly, it may happen that all paths are delayed and no packets after p_1 are received. However, in this case either there is a serious delay on all paths (in which case even a single-flow TCP would suffer a timeout), or p_1 is the last packet in the window (which does not happen very often).

Secondly, packet p_1 maybe lost due to a wireless error, in which case we are unneccessary delaying other packets. To that end we introduce an additional mechanism, explained in Section 5, to detect packet losses when possible. Once packet p_1 is declared lost, we continue as if p_1 has been correctly delivered.

The delayed reordering procedure cannot completely eliminate timeouts or triple duplicate ACKs when one path is significantly delayed or lossy. However, it significantly reduces the rate of these undesirable events to the point that we can efficiently explore multiple paths and outperform single-path TCP in many cases, as discussed in Section 6.

5. HORIZON ARCHITECTURE

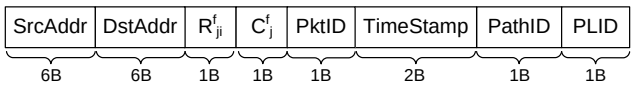

Figure 2: Structure of *Horizon* packet header (B - byte, b - bit). Its total length is 19 bytes. Different fields are explained in Section 5.1.

We have implemented *Horizon* at the layer 2.5 as a user-space daemon within VRR routing protocol [3] on Windows XP (although the architecture is in no way tied to a particular OS). VRR is an NDIS driver between MAC and IP layer which intercepts and can reinsert a packet to the MAC and to the IP layer. It then forwards them to *Horizon* for processing. *Horizon* also periodically queries VRR to obtain the routing tables. When preparing a packet for transmission, *Horizon* picks the best route according to its own packet forwarding mechanism (discussed in Section 3) and prepares the packet with a destination's Ethernet MAC address. It then transmits the packet directly to wireless MAC. At a destination, packets are delivered to IP according to the delayed-reordering mechanism, described in Section 4. All state information in *Horizon* is soft, which enables it to react to topology and traffic changes. *Horizon* does not provide any guarantees on delivery; this is left to the upper layers. It also doesn't support packet fragmentation. As a common wisdom, we turn off RTS/CTS for performance. We next discuss different important implementation aspects of our design.

5.1 Header Structure

Horizon adds its own packet header (illustrated in Figure 2) between Ethernet and IP headers. Fields *SrcAddr* and *DstAddr* are packet's flow source and destination MAC addresses. They are used to identify the flow. We do not perform deep packet inspection of TCP header to determine the flow a packet belongs to. If there are several TCP flows between the same source and destination, they are treated as the same *Horizon* flow to simplify the design. Each packet in a flow is labelled sequentially with 1-byte *PktID* (wrapped when neccessary), and the destination can rearrange the packets at the *Horizon* layer according to their PktIDs before delivering them to the upper layers. Also, on each link, a packet is labelled sequentially with 1-byte *PLID*. *PLID* is packet sequence number per link and per flow. Link destination can detect packets losses in the MAC by detecting a missing *PLID*. We also use *TimeStamp* header field to estimate one-way delays (to signal $t_s(p)$, as explained in Section 4).

Horizon also uses *PathID* header field to track paths packets have used. Upon initialization, each node picks an 1-byte random ID. Initially, *PathID* of a packet is set to the random ID of the source node. Each node that forwards a packet XORs its random ID with the existing *PathID* value in the packet header and stores it back in *PathID*. It is then highly probable that the *PathID* of the packets traversing different paths will differ, whereas packet traversing the same path will always have the same *PathID*. Finally, R^f_{ji} and C^f_j are used to determine prices; see Section 5.2.1.

5.2 Pricing and Forwarding

5.2.1 Packet Transmissions and Acknowledgements

Path qualities are estimated through queue sizes. *Horizon* manages its own per-flow queues but it has no control over the packets queued at the MAC level, since it is above the MAC layer. Firstly, it is difficult to track the number of packets in the MAC-layer queue without exploring 802.11 driver's architecture and thus being attached to a specific hardware. Also, it is not clear if we can assume that *Horizon* is the only layer bound to the MAC layer (and count the MAC egress packets). Finally, since *Horizon* is currently implemented in the OS user space, frequently switching context to query 802.11 driver's queue decreases performance.

The main question is the rate of forwarding packets to the MAC layer? If *Horizon* forwards all the packets immediately, its queue will be empty and it will have no means to estimate path quality. Also, excessive packets will be dropped by the MAC, as may happen in the conventional network stack. Clearly, it should try to keep packets in its own queue and occasionally forward them to the MAC. However, having no packet in the MAC queue when the MAC is ready to transmit will drop the link utilization.

In order to determine when *Horizon* should forward packets to the MAC, we introduce a concept of responsibility for packets. A node is responsible for a packet until someone else takes over the responsibility or until the packet is dropped due to MAC loss. We divide the total packets in the custody of node i in three groups: (I) the packets queued at *Horizon* at node i, (II) packets queued at MAC level of node i, and (III) packets already transmitted by node i but not acknowledged.

Ideally, we would like to minimize the number of packets in group II. Instead, we will try to control the number of packets in groups II and III. To that end, we shall use explicit acknowledgments at *Horizon* level to transfer responsabilities for packets. Node i keeps track of S^f_{ij}, the *PLID* of the last packet of flow f sent from i to j. Node j acknowledges to node i a successful reception by sending R^f_{ji}, the *PLID* of the last packet of flow f received from i. That means node j relieves node i responsability for all packets of flow f transmitted from i to j with *PLID* lower or equal to R^f_{ji}.

Node i needs to determine the number of packets that are in its custody, P^f_i. Value P^f_i can be interpreted as the number of locally queued packet of flow f and it is used to determine the price, as explained in Section 3. Node i knows S^f_{ij} and it also occasionally receives R^f_{ji} in a refreshment packet (as described in Section 5.2.2). Then, the total number of packets in transition between *Horizon* layers at node i and j (group II and III) is $S^f_{ij} - R^f_{ji}$. These packets are

still the responsibility of node i, and it counts them as not yet delivered.

Let Q_i^f be the total number of packets queued at node i at *Horizon* level for flow f (group I). Then the total number of packets in the custody of node i for flow f is $P_i^f = Q_i^f + \sum_{j \in \mathcal{D}(f)}(S_{ij}^f - R_{ji}^f)$. This is the number of packets node i reports to its upstream nodes (by definition we set $P_{d(f)}^f = 0$ where $d(f)$ is the destination of flow f).

Finally, in order to minimize the number of packets in group II and III but avoid starving the MAC layer, we allow node i to transmit a packet from flow f to node j only if $S_{ij}^f - R_{ji}^f$ is less than a back-pressure threshold T. We discuss the choice of T below.

5.2.2 Signalling Updates

Downstream nodes need to signal to their upstream peers their queue status for each flow. This is done by either appending these information to a packet in the reverse direction or sending a dedicated 'refreshment' packets. A refreshment packet from node j to node i and flow f contains C_j^f and R_{ji}^f. It is a short packet that contains no payload (fields C_j^f and R_{ji}^f are already included in the header).

The values of R_{ji}^f and C_j^f are updates whenever a new packet is received. However, the value of C_j^f changes more often - it also changes when node j receives new refreshments. It is important that node j signals updates to node i frequently enough to reflect these changes, but not too frequently to avoid creating extra contention in the wireless medium.

The choice of refreshment frequency affects the choice of back-pressure threshold T. The optimal value of T also depends on the MAC layer characteristics. If T is too small, we risk underutilizing the link by not supplying enough packet. On the other hand, if T is too large, we may disrupt load-balancing by sending extra packets on low-quality routes.

We experimentally find that the best performance is achieved when node j sends refreshments about flow f to its upstream neighbors approximately after every 5 consecutve changes of C_j^f and when $T = 10 - 15$. The study of this trade-off is left for future work.

In the case when a direct and a reverse flow's paths coincide, C_j^f and R_{ji}^f are piggybacked in the header of a packet from the same flow traveling in the reverse direction (e.g. TCP ACK packets). Although a routing protocol does not have to guarantee that the direct and the return path will coincide, we empirically verify that this significantly reduces the *Horizon* signalling overhead.

5.3 Interaction with TCP

Other issues that affect TCP performance include: how to detect wireless losses and thus avoid unneccessary delays in delivery, how to correctly dimension the utility function and how to signal congestion.

5.3.1 Detecting Wireless Losses

We can conclude that packet p_1 of flow f is missing due to a wireless loss is if we have received packets at the destination of sd with *PktID* greater than p_1 on all of the learned path from flow sd (paths are learned through *PathID*; see Section 5.1). Since ordering on each path is guaranted, we can undoubtedly conclude that packet p_1 is lost, and continue delivering. Although this approach will not detect all losses, we experimentally observe that it significantly decrease the unneccessary reordering.

Additionaly, if a packet is small (less than 60 B), we conclude that it consists of TCP ACK or a TCP window probe. In that case we can deliver it immediately as a reception of such a packet will not generate any further ACKs and there are no triple duplicate ACK problems. This heuristics helps us further decrease the reordering delays.

5.3.2 Dimensioning Utility Function

We next discuss the dimensioning of factor K in the utility function, defined in Section 4.1. Large K results in small $C_{s(f)}^f$ which in turn implies smaller queues. However, if there are very few packets in the queues, our path estimates will not be accurate which will impact the performance. We empirically select $K = 90$ such that $C_{s(f)}^f \approx 5$ for window size $B_f = 40$ packets ($\approx 64kB$). We do not signal congestion if the window is smaller than 5 to avoid clogging a fast flow. We also do not let $C_{s(f)}^f$ grow over 100, regardless of the window size, to avoid excessive delays ($C_{s(f)}^f = 100$ corresponds to $B_f = 9$).

The optimal choice of K is prone to a potential scaling issues and there might be a need for a dynamic adaptation of K as a function of path length. However, we do not forsee mesh networks dramatically increase in size and we leave this issue for a future work.

5.3.3 Signalling Congestion

We constantly estimate the window size B_f by estimating y_f and RTT_f. Once we detect that B_f is larger than its optimal value $K/\sqrt{C_{s(f)}^f}$, we need to signal congestion to TCP. There are several possible ways to do this. A universal way is to drop excessive packet. In addition to TCP, this works well with other transport protocols, such as UDP. However, it may cause unneccessary packet losses. Instead, we propose an explicit congestion notification for TCP. Since Windows XP TCP stack does not support ECN, we send a congestion indicator by sending a fake triple duplicate ACK. For that purpose, we keep a track of the last packet delivered to the IP at the source of flow f (a packet that carried an ACK for flow f). Once a congestion is detected, we deliver the same packet again three times. TCP will treat it as a triple duplicate ACK and will halve the congestion window.

5.4 Simple Routing Protocol

Performance of *Horizon* depends to large extent on qual-

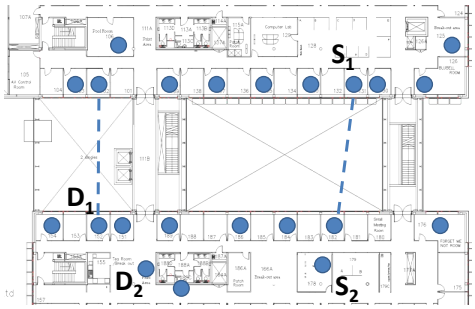

Figure 3: Floorplan of the testbed. Dots represent nodes' locations. The area in the middle is an atrium. Only the nodes connected with dashed lines can communicate across the atrium.

ity of routes. Our goal is to analyze *Horizon* and abstract the routing issues. We use VRR to obtain link-state information. We use static anchors, hand-picked for our test-bed (as explained in Section 6.1) to obtain satisfactory disjoint paths. Due to topology constraints of the current test-bed, we use only two paths per flow. We leave the questions on how to derive the optimal routing protocol and how many paths to use for the future work.

6. PERFORMANCE EVALUATION

6.1 Test-bed Topology

We evaluated the performance of *Horizon* on a wireless mesh testbed, comprising 22 nodes on one floor of our building, shown in Figure 3. Each node is a PC with one or two 802.11a cards equipped with omni-directional antennas. There are 2 pairs of multi-homed nodes(connected by dashed-lines in the figure), which act as bridges. The empty area in the middle of the building is an atrium — only bridges can talk accross the atrium as they have antennas mounted on the outside of the office window (bridge communication is denoted with the dashed lines). Each bridge is equipped with 2 NICs each, one for the atrium and one inside. Bridges also serve as static 'anchors' (Section 5.4). Each pair of bridges uses a different frequency, and they do not interfere with each other. All nodes in each half of the building use one frequency (including the bridges' interior NICs), with a different frequency used in the two halves.

6.2 Performance Metrics

Our test-bed has a form of a ring, and offers two disjoint paths between any node-pairs. When the number of flows competing for scarse wireless resource is small, *Horizon* increases the total throughput of the system by utilizing resources more efficiently, exploiting multiple paths.

On a contrary, when a large number of flows are active in the system, there is a high chance that each bridge-link is being used. By virtue of TCP's behavior, the flows will attempt to fully use the two bridge links, and we cannot further increase the total throughput. The problem then becomes one of fairness between flows.

There are several ways to quantify the trade-off between the efficiency and fairness. With our utility maximization framework, (Section 3) , the most natural one for us is the **total system utility** $\sum_f U_f(y_f)$. However, utility is not an intuitive performance measure. Therefore we also use four other performance metrics: the **rate of the maximum flow** $\max_f y_f$, the **rate of the minimum flow** $\min_f y_f$, the **mean flow rate flow** $\frac{1}{|\mathcal{F}|}\sum_f y_f$ and the **Jain's fairness index** [16] $\left(\sum_f y_f\right)^2 / \left(n \sum_f y_f^2\right)$.

6.3 Illustrative Examples

We first consider an example from Figure 3 with two flows: $S_1 - D_1$ and $S_2 - D_2$. Flow $S_1 - D_1$ can use two paths, one that interferes with $S_2 - D_2$ and the other which does not. We let flow $S_2 - D_2$ run for 20s and $S_1 - D_1$ for 40s. The results are illustrated in Figure 4.

First consider what happens with normal single-path TCP. Suppose that flow $S_1 - D_1$ uses a path that overlaps with $S_2 - D_2$. During the first 20s, when both flows are active, the average rate for $S_2 - D_2$ is about 600 kB/s, and 190 kB/s for flow $S_1 - D_1$. Subsequently, when flow $S_1 - D_1$ is alone, its rate increases to about 340 kB/s.

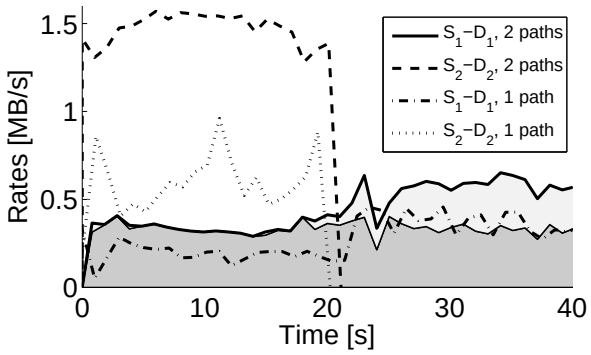

Figure 4: An example of load balancing with two flows: flows $S_1 - D_1$ and $S_2 - D_2$ (see Figure 3) compete for resources. Flow $S_2 - D_2$ finishes after approx 20s. The light gray area is the rate of flow $S_1 - D_1$ over the path shared with $S_2 - D_2$, dark gray is the rate over a disjoint path.

Now consider the same scenario using *Horizon* . Note that *Horizon* uses the same routing protocol as for single-path routing, hence the default path for $S_1 - D_1$ is also the one that overlaps with $S_2 - D_2$. However, when two flows start transferring data, *Horizon* is able to detect the congestion on one of the paths and transfer almost all of the packets from flow $S_1 - D_1$ over the non-congested path (the dark-shaded area in Figure 4 showing the traffic over the non-congested path). This approximately doubles the rates for both flows because of a better resource utilization, achieveing 1.4 MB/s for $S_2 - D_2$ and 330 kB/s for $D_1 - S_1$. In this example *Horizon* effectively acts as an ideal routing protocol, selecting the best of the proposed paths in a dynamic and timely

manner.

When flow $S_2 - D_2$ finishes, *Horizon* detects the other path is no longer congested, and starts using that as well. It balances traffic over both paths in such way that the prices of both paths are approximately the same. Again, the rate of $S_1 - D_1$ is almost doubled compared to the previous case (620 kB/s), since both paths are (fully) used.

6.4 Random Flows

We now describe the performance of randomly selected, concurrent flows on our testbed. Each experimental run represents randomly selected pairs of flows (source-destination pairs). For each set of source-destination pairs we repeated the measurements 10 times. We ran sets of experiments with 1, 2, 4 and 8 flows. Our test-bed exhibited very poor performance with more than 8 bandwidth-hungry concurrent flow, with and without multi-path, due to interference, hidden-terminal problems and TCP timeouts, and yielded no meaningful conclusions.

The samples are not ergodic and do not follow any obvious distribution, hence it is difficult to calculate confidence intervals. Instead, whenever we plot flow rates with error bars (Figure 5 and Figure 6), the error bars depict the minimal, maximal and mean values from the set of repeated measurements.

6.4.1 *Single Flow Case*

Figure 5(a) shows the throughput of a single flow for different randomly selected source and destination pairs .

We can classify the resulting allocations into 5 groups. The distinction among different groups is roughly shown in Figure 5(a). In group I there are flows whose performance drops due to *Horizon* scheduling inefficiencies, discussed in Section 3.6. For all of these flows, both the source and the destination contain a single NIC (but not all nodes with single NIC experience this inefficiency; some are in group III). Group II are short flows that do not use multiple paths. *Horizon* exhibits slightly worse performance (around 5%) than single-path TCP due to the protocol and CPU overheads.

Group III are flows in which both the source and the destination contain a single NIC, but which do not suffer the scheduling problems of group I. Observe that *Horizon* appears to slightly improves the average rate (by 10%-20%). Group IV are flows in which either the source or the destination (but not both) contain two NICs. The average rate improves by 20%-50%. Finally, group V are the flows in which both the source and the destination are multi-homed. Here we see the largest performance improvement of *Horizon* , which goes as high as 100%. Flows with at least a source or a destination having two NICs are depicted in Figure 5(b).

We attribute the performance degradation of group I to 802.11 MAC scheduling, and verify it empirically. When we introduce multi-homed nodes, we eliminate these problems. *Horizon* then significantly improves the performance through using multi-path. This implies that the more a mesh network is designed to exploit multi-channel capabilities, the greater the performance improvement for a single flow. It remains as future work to test this hypothesis with *Horizon* in a test-bed with more multi-homed nodes. Furthermore, as we shall see in the following sections, self-contention effects become less important with multiple flows as the contention among flows becomes substantial.

6.4.2 *Two Flows Case*

We now described experiments with two concurrent flows. The achieved rates are depicted in Figure 6. The relative improvements of different performance metrics are shown in Figure 7.

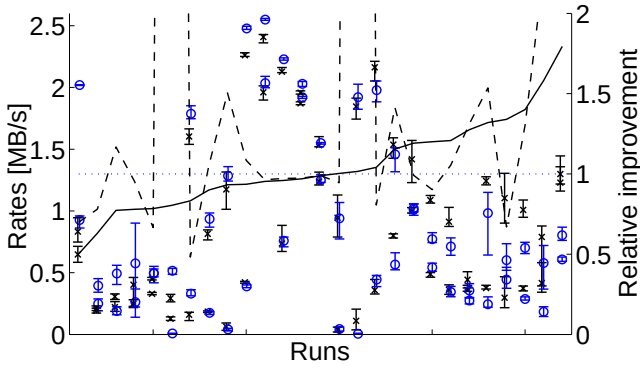

Figure 6: Experiments with 2 flows. We plot the achieved rates of the two flows for both *Horizon* and single-path TCP in different experiments (runs). Solid line represents the relative improvement and dashed line the improvement of the worst flow (points missing from the graph represent 15-22 times improvements).

From Figure 7 we see that in about 70% of cases the total utility has increased, the fairness index and the performance of the worst flow is increased in about 50% of the runs and the sum of the flows' rates (i.e. mean rate) has increased in approximately 30% of the cases. We again see the problems of inefficient scheduling, which result in approximately 30% of the flows having decreased utility.

From Figure 6 we see that in some cases fairness was improved at the expense of total rate. However, in the runs on the right side of Figure 6 we see that we have improved both the total rate (mean rate) and the fairness (the rate of weakest flow). As discussed in Section 6.2, this is possible since in many cases the two flows will be obliged to share the same paths with single-path TCP, whereas *Horizon* will balance them on two paths and improve the resource utilization. In summary, in most of the cases we improve fairness and in some cases we improve both fairness and total throughput.

6.4.3 *Cases with more than two flows*

The results for experiments with 4 and 8 flows are depicted in Figure 7. Confidence intervals are of the same order as for experiments with 1 and 2 flows, but omitted for clarity. From Figure 7 we can conclude that *Horizon* in most of the

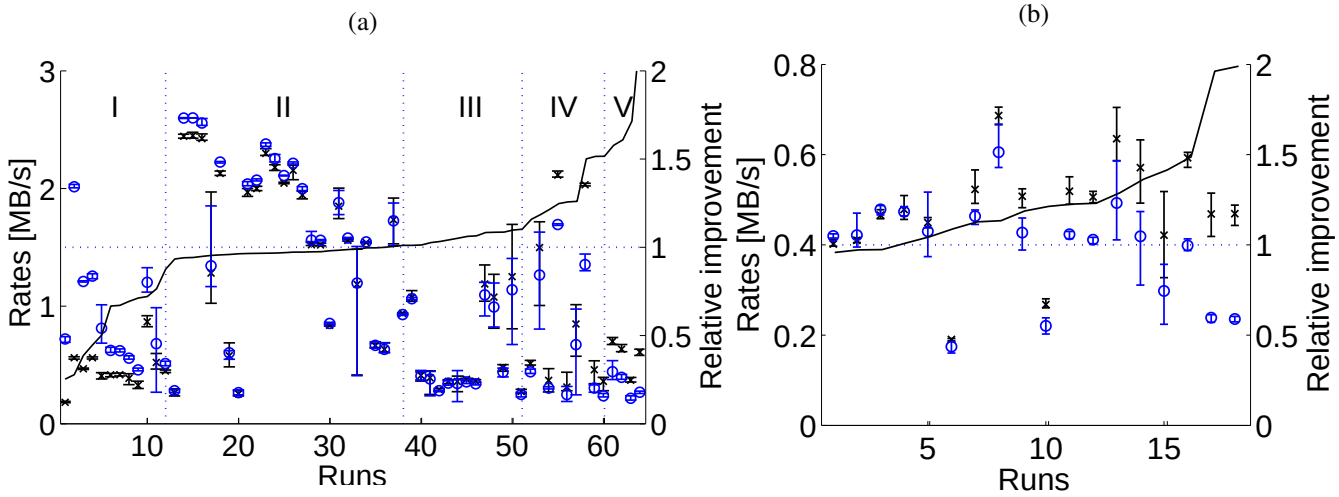

Figure 5: Single-flow case: In (a) we randomly select flows source-destination pairs among all available nodes. In (b), subset of (a)with at least a source or a destination that have two NICs (groups IV and V). Black crosses represent rates (in kB/s) achieved using *Horizon* , blue circles using single-path TCP. Error bars are explained in Section 6.4. Black line is the relative improvement of *Horizon* over single-path TCP (corresponding y axis is on the right).

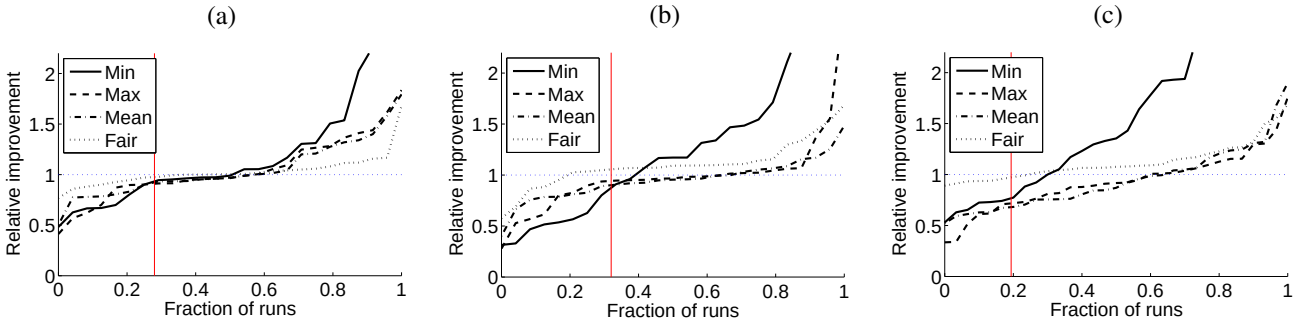

Figure 7: CDF of the relative improvement of the minimum, mean, maximum rate and the fairness index of *Horizon* over the single-path TCP for the experiments with 2 flows (a), 4 flows (b) and 8 flows (c). Confidence intervals are removed for clarity. Note that the points on different curves with the same x-value do not correspond to the same experiment. The experiments on the left of the vertical red line have lower utility in the case of *Horizon* than in the case of the single-path TCP. The experiments on the right improve the utility with *Horizon* .

cases improves the system utility, and the improvement is more visible when the number of flows grow large. We also see that the fairness is improved and that the fairness index and the performance of the worst flow improve significantly. As expected, this is traded-off against the sum of rates (mean rate) and the performance of the best flow.

Notice the 'flat' part of the mean rates curves. This is caused by most of the randomly selected flows being short flow, which all experience similar performance. In many cases the short flows contribute significantly to the overall throughput. As the number of flows increases, the level of the flat part decreases. This means as the number of flows, and opportunities to trade increase, we tend to trade more the performance of the short flows to improve that of the long flows, hence we improve fairness.

6.4.4 *Load-balancing*

Finally, we investigate load-balancing properties. We want to show that *Horizon* not only selects one optimal path per flow, but in many cases it actively balances traffic over all available paths (sends more than 5% of traffic over both paths). We verify that in at least 60% of all the experiments with 2 flows at least one flow actively balances traffic. Similarly, in at least 70% of the experiments with 4 flows at least one flow actively balances traffic, and in at least 20% of the

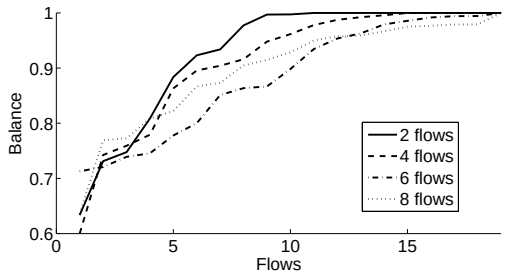

Figure 8: Fraction of the total traffic sent on one of the two paths

experiments two flows actively balance traffic. The CDF of the distribution of traffic over path for flows that use multiple paths is given in the graph in Figure 8, and the traffic split goes up to 60:40 for certain flows.

7. RELATED WORK

One of the first analysis of TCP over a single wireless link is given in [1]. An experimental analysis of TCP performance in wireless multi-hop networks is given in [10]. They find that RTS/CTS should be switched off for performance and that one should not increase retransmission counter too large as bad links will get too much transmission opportunities. Sender-side modification of single-path TCP is proposed in [13]. It estimates bandwidth and adapts the window to the estimated value upon triple ACK.

Some practical results on flow control and single-path routing in mesh networks are given in [7, 11, 15]. In [11], where back-pressure is used in a different manner to prevent congestion at MAC layer. It requires MAC-layer modifications and is verified by simulations only. Similar results that prevent MAC-layer contention using flow control are presented in [7, 15].

Multi-path routing in sensor networks has been proposed in [17]. It requires coordinates and provides disjoint paths based on the topology. It has its own flow control scheme, doesn't use TCP. Multi-path routing with TCP has been proposed in [21]. It measures RTT on each path and sends traffic proportionally to RTT. Tested by simulations only and in lightly-loaded network (no reordering).

There is a long history of multi-path routing and TCP in wired networks. Some of the examples are [4, 9]. In [9] a flowlet level balancing is proposed to avoid reordering effects. [4] proposes a multi-path load balancing scheme under predefined weights that minimize average packet delays. How to prevent reordering and timeouts in TCP using DSACK is discussed in [22].

Originally, back-pressure scheduling has been proposed in [20], where it is shown that this scheduling can stabilize a network whenever possible. This paper generated a whole new direction of research on the jointly optimal scheduling, routing and flow control in wireless networking (c.f. [5,6,12, 14]); a comprehensive survey can be found in [8].

We did not compare *Horizon* numerically to the algorithms from [7, 11, 15, 17, 21] because these algorithms either do not satisfy all our design requirements or there is no available implementation. To our knowledge, we are the first one to implement back-pressure ideas in a system design.

8. CONCLUSIONS

We presented a novel system architecture *Horizon* for load-balancing and multi-path routing in wireless mesh networks. Our design guarantees efficient use of available resources, fairness among cometeing flows, and it works with unmodified 802.11 MAC and TCP/IP.

We started from adopting the recent theoretical ideas about back-pressure scheduling and utility maximization. We proposed novel solutions to several problems that arise in practical applications of these algorithms and we demonstrated that it is indeed possible to use these ideas in practice. Unlike previously proposed system architectures in this space, *Horizon* can guarantee some network-wide performance characteristics, works with unmodified networking networking stack and the claims are evaluated and confirmed in a real-world test bed (and not by simulations).

Acknowledgment

The authors are grateful to Greg O'Shea and the rest of the VRR crowd for the fully functional test-bed and all the help and advices.

9. REFERENCES

[1] H. Balakrishnan, V. Padmanabhan, S. Seshan, and R. Katz. A comparison of mechanisms for improving TCP performance over wireless links. *IEEE/ACM Transactions on Networking*, 5(6):756–769, 1997.

[2] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[3] M. Caesar, M. Castro, E. B. Nightingale, G. O'Shea, and A. Rowstron. Virtual ring routing: network routing inspired by DHTs. In *ACM SIGCOMM*, 2006.

[4] C. Cetinkaya and E. Knightly. Opportunistic traffic scheduling over multiple network paths. In *Proceedings of INFOCOM*, 2004.

[5] M. Chen, S. Low, M. Chiang, and J. Doyle. Cross-layer congestion control, routing and scheduling design in ad hoc wireless networks. In *INFOCOM*, 2006.

[6] A. Eryilmaz and R. Srikant. Joint congestion control, routing and mac for stability and fairness in wireless networks. *IEEE Journal on Selected Areas in Communications*, 24(8):1514–1524, August 2006.

[7] Z. Fu, P. Zerfos, H. Luo, S. Lu, L. Zhang, and M. Gerla. The impact of multihop wireless channel on TCP throughput and loss. In *INFOCOM*, 2003.

[8] L. Georgiadis, M. Neely, and L. Tassiulas. Resource allocation and cross-layer control in wireless networks. *Foundations and Trends in Networking*, 1(1):1–144, 2006.

[9] S. Kandula, D. Katabi, S. Sinha, and A. Berger. Dynamic load balancing without packet reordering. *ACM SIGCOMM Computer Communication Review*, 37(2), April 2007.

[10] V. Kawadia and P. R. Kumar. Experimental investigations into TCP performance over wireless multihop networks. In *Wksp on Exp. approaches to wireless network design and analysis*, 2005.

[11] C. Lim, H. Luo, and C.-H. Choi. RAIN: A reliable wireless network architecture. In *Proceedings of ICNP '06*, pages 228–237, 2006.

[12] X. Lin and N. Shroff. Joint rate control and scheduling in multihop wireless networks. In *43rd IEEE CDC*, 2004.

[13] S. Mascolo, C. Casetti, M. Gerla, M. Y. Sanadidi, and R. Wang. TCP Westwood: Bandwidth estimation for enhanced transport over wireless links. In *MCN'01*, pages 287–297, 2001.

[14] M. Neely, E. Modiano, and C. Rohrs. Dynamic power allocation and routing for time-varying wireless networks. *IEEE Journal on Selected Areas in Communications*, 23(1):89–103, January 2005.

[15] C. Pazos, J. Sanchez-Agrelo, and M. Gerla. Using back-pressure to improve TCP performance with many flows. In *Proceedings of INFOCOM*, pages 431–438, 1999.

[16] L. Peterson and B. Davie. *Computer Networks: A Systems Approach*. Academic Press, 2000.

[17] L. Popa, C. Raiciu, I. Stoica, and D. Rosenblum. Reducing congestion effects by multipath routing in wireless networks. In *ICNP'06*, pages 96–105. IEEE, 2006.

[18] R. Srikant. *The Mathematics of Internet Congestion Control*. Birkhauser, 2004.

[19] R. Stevens. *TCP/IP Illustrated: Protocols*. Addison Wesley, 1994.

[20] L. Tassiulas and A. Ephremides. Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks. *IEEE Trans. on Automatic Control*, 37(12), 1992.

[21] Z. Ye, S. V. Krishnamurthy, and T. S. K. Effects of multipath routing on TCP performance in ad hoc networks. In *Proc. of IEEE GLOBECOM*, 2004.

[22] M. Zhang, B. Karp, S. Floyd, and L. Peterson. RR-TCP: a reordering-robust TCP with DSACK. In *IEEE ICNP*, 2003.