One Flood to Route Them All: Ultra-fast Convergecast of Concurrent Flows over UWB

Matteo Trobinger, Davide Vecchia, Diego Lobba, Timofei Istomin, Gian Pietro Picco*
{matteo.trobinger,davide.vecchia,diego.lobba,timofei.istomin,gianpietro.picco}@unitn.it
University of Trento, Italy

ABSTRACT

Concurrent transmissions (CTX) enable low latency, high reliability, and energy efficiency. Nevertheless, existing protocols typically exploit CTX via the Glossy system, whose fixed-length network-wide floods are *entirely* dedicated to disseminating a *single* packet.

In contrast, the system we present here, Weaver, enables *concurrent* dissemination towards a receiver of *different packets* from *multiple senders* in a *single, self-terminating*, network-wide flood.

The protocol is generally applicable to any radio supporting CTX; the prototype targets ultra-wideband (UWB), for which a reference network stack is largely missing. Our modular design separates the low-level mechanics of CTX from their higher-level orchestration in Weaver. Other researchers can easily experiment with alternate designs via our open-source implementation, which includes a reusable component estimating UWB energy consumption.

Our analytical model and testbed experiments confirm that Weaver disseminates concurrent flows significantly faster and more efficiently than state-of-the-art Glossy-based protocols while achieving higher reliability and resilience to topology changes.

CCS CONCEPTS

• **Networks → Network protocol design**.

KEYWORDS

Concurrent transmissions, ultra-wideband, low-power wireless.

ACM Reference Format:
Matteo Trobinger, Davide Vecchia, Diego Lobba, Timofei Istomin, Gian Pietro Picco. 2020. One Flood to Route Them All: Ultra-fast Convergecast of Concurrent Flows over UWB. In *The 18th ACM Conference on Embedded Networked Sensor Systems (SenSys '20), November 16–19, 2020, Virtual Event, Japan.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3384419.3430715

1 INTRODUCTION

In the last decade, concurrent transmissions (CTX) have catalysed the attention of the low-power wireless community. The term refers to the fact that tightly-synchronized simultaneous transmissions do not necessarily result in a collision; instead, under some conditions determined by the underlying PHY radio layer (§2), one of the concurrent packets is received with very high probability.

Motivation. This phenomenon was exploited in IEEE 802.15.4 narrowband where, popularized by the Glossy system [12], it rapidly became a state-of-the-art asset in designing protocols supporting various traffic patterns [11, 17, 21]. Recent work has shown that the same principle is applicable to other radios, notably including Bluetooth Low Energy (BLE) [26] and ultra-wideband (UWB) [20, 23, 29].

This surge of interest is motivated by the fact that protocols based on CTX enable unprecedented performance by achieving at once near-perfect reliability and very low latency and energy consumption. Nevertheless, most of these protocols are built atop the network-wide flooding offered by Glossy, by simply scheduling its floods in different ways depending on the traffic pattern and goals at hand. In other words, Glossy is used to a large extent as a *monolithic blackbox*, with little or no modifications by higher-layer protocols. This is a reasonable design decision enabling faster and reliable prototyping, given the system-level complexity of Glossy. On the other hand, this decision stifles the exploration of alternate, finer-grained designs that *directly and individually* exploit CTX, free from the rigid, predefined scheduling strategy of Glossy.

Exploiting the full potential of CTX: Weaver. This second approach is precisely the one we follow in this paper, in which we retain network-wide flooding as the main communication mechanism, but fundamentally change its purpose and operation.

A Glossy flood is *entirely* dedicated to disseminating a *single* packet across the entire network. A global scheduling of transmission (TX) and reception (RX) slots, whose redundancy is governed by the user-defined parameter N, achieves the aforementioned excellent performance. Nevertheless, flooding is still a wasteful, network-wide operation, exacerbated by the fact that reliability is directly tied to the redundancy factor N; the higher its value, the higher the number of times a packet is (concurrently) retransmitted and, therefore, the higher the (network-wide) energy consumption. Crucially, the value of N, and hence the duration of the flood, is fixed before execution, therefore intrinsically prone to over- or under-provisioning, hampering lifetime and reliability, respectively.

In contrast, the system we describe here, called Weaver, is expressly designed to *concurrently disseminate towards a receiver different packets from multiple senders in a single, self-terminating, network-wide flood*, significantly improving on latency, reliability, and energy consumption w.r.t. Glossy-based systems.

Weaver achieves these goals with several mechanisms, each built directly atop individual CTX. As in all Glossy-based systems, Weaver alternates (short) periods executing its network-wide flood with (long) periods of inactivity, all implicitly time-synchronized by system operation. Each node, including the sink, executes a

*The first two authors contributed equally to the research in this paper.

SenSys '20, November 16–19, 2020, Virtual Event, Japan
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-7590-0/20/11…$15.00
https://doi.org/10.1145/3384419.3430715

time-slotted sequence formed by a TX slot followed by *two* RX slots, that repeats until the flood self-terminates. Adding an extra RX slot to the TX-RX scheme of Glossy may seem a minor change; yet, it is crucial to unlock significant performance gains. Indeed, the resulting 3-slot structure, combined with the propagation of an initial message from the sink, staggers the (concurrent) TX and RX of nodes at different hop counts from the sink, enabling multiple flows to co-exist within the same flood without disrupting each other. Further, it enables Weaver to exploit a combination of *local*, 1-hop broadcast acknowledgments and *global*, sink-initiated ones that, together, adaptively *i)* suppress unnecessary packet propagation or, on the contrary *ii)* trigger retransmission of packets that have been lost, therefore decreasing energy consumption and increasing reliability w.r.t. the fixed redundancy of Glossy-based approaches.

Goals, methodology, contributions. We discuss the design rationale and goals for Weaver (§3) and offer an analytical model confirming the intrinsically superior performance achieved w.r.t. Glossy-based state-of-the-art representatives, before delving into a more in-depth illustration of protocol details (§4).

Systems based on CTX are notoriously complex. This, however, is to a large extent a relic of a past when the lack of proper hardware primitives required complex designs yielding timing guarantees. Nowadays, many radio transceivers offer rich primitives notably including the ability to schedule transmissions accurately, including the Decawave DW1000 ultra-wideband (UWB) radio [7] we focus on in this paper. Other works [20, 23] have shown that Glossy-based protocols can be effectively supported by its powerful features.

Moreover, we observe that a staple communication stack, and in particular a convergecast one, is currently missing for UWB, in stark contrast with the plethora of protocols resulting from more than a decade of work on IEEE 802.15.4 narrowband wireless sensor networks. By providing a fast and efficient data collection embodied by our Weaver prototype we aim at fostering adoption of UWB also for sensing and communication, besides ranging and localization.

Once the leap is made from coarse-grained, rigid Glossy floods to finer-grained alternatives based on individual CTX, many solutions are possible, catering for different requirements. Our *modular* implementation (§5) sharply decouples the mechanics of accurately scheduling TX and RX slots, delegated to a Time Slot Manager (TSM) component, from their higher-level orchestration in Weaver, which can be easily replaced by alternate designs. A component estimating energy consumption is also provided. We release these reusable components as open source [1] contributing to further developments in the fast-growing UWB research community.

We evaluate Weaver in a 36-node testbed at our premises. We first analyze the impact of key design decisions with dedicated experiments. Next, we compare directly the performance of Weaver against Crystal [17, 18] a state-of-the-art Glossy-based convergecast protocol with an UWB implementation [23]. Our results confirm the trends observed in the analytical model, e.g., showing that Weaver can deliver at the sink 30 concurrent flows in ∼100 ms, achieving a reduction of ∼70% in both latency and energy consumption w.r.t. Crystal while achieving near-perfect reliability due to the lower contention induced by the finer-grained, adaptive use of CTX. Moreover, the ultra-fast dissemination achieved by Weaver, along with the inherent redundancy offered by CTX, makes our system resilient to topology changes, e.g., induced by mobility.

Finally, although our Weaver prototype targets UWB, its protocol design does not rely on features specific of this PHY layer. Therefore, it can be applied to other radios, amplifying the impact and contribution outlined here and pushing the envelope of what CTX can achieve in low-power wireless communications at large. We concisely discuss these and other follow-up opportunities (§7) before ending the paper (§8) with brief concluding remarks.

2 BACKGROUND AND RELATED WORK

We offer the necessary background on CTX in narrowband and UWB, along with a concise survey of the most relevant approaches.

2.1 CTX in IEEE 802.15.4 narrowband

In these radios, where the concept was first developed [4], CTX rely on two PHY-level phenomena occurring when senders simultaneously transmit towards the same receiver on the same RF channel. *Constructive interference* is possible when copies of the *same* packet arrive within 0.5 μs, i.e., the duration of a bit (chip) in the DSSS modulation; the union of concurrent, identical signals actually improves the reliability of packet RX. The *capture effect* is possible when *different* packets arrive within 160 μs, i.e., the duration of the synchronization header, enabling the radio to switch RX from a signal to another; one of the packets is likely received, depending on the number of senders and their relative signal strength.

Glossy as a reusable building block. Glossy [12] was the first to exploit CTX into a reliable, efficient, and publicly-available system providing network flooding and time synchronization. Several others exploited the low-latency, high-reliability, low-consumption, network-wide flooding of Glossy as a building block for higher-level abstractions. LWB [11] supports different traffic flows (many-to-one, one-to-many, one-to-one) by properly scheduling them as individual Glossy floods from a single initiator. Crystal [17, 18] supports many-to-one convergecast via phases in which Glossy floods from multiple initiators compete, followed by others in which the sink alone has the opportunity to acknowledge the received packet. Other systems [3, 27, 28, 31] explore variants of these concepts.

Reusability vs. degrees of freedom. Interestingly, in all these systems Glossy is reused as a *monolithic blackbox*, with little or no modification. Indeed, only few systems make relatively small modifications to Glossy that, however, are not geared to change its core functionality, rather to improve its performance, e.g., increasing reliability via channel hopping [25] and/or reducing latency [2, 22].

The direct reusability of Glossy actually fueled research on CTX, enabling researchers to experiment with new protocols while avoiding the intricacies of the CTX implementation. Nevertheless, at the same time it also fossilized research on CTX to a large extent.

Indeed, a Glossy flood is entirely dedicated to disseminate a *single* packet from a *single* initiator, with the intent to exploit constructive interference among forwarding nodes. LWB and others (e.g., [28, 31]) rely directly on this feature. Crystal and others (e.g., [27]) push Glossy further by having *multiple* initiators compete within the same flood, relying on the capture effect; however, the final outcome is still a single packet from only one of the initiators.

Weaver: Back to individual CTX. In contrast, we take a significantly finer-grained perspective and free ourselves from the mechanics of Glossy, exploiting CTX directly and individually.

Only few systems hitherto resorted to a similar approach, and always to support many-to-many communication. Chaos [21] realizes network-wide aggregation of data from multiple initiators via competing floods. Mixer [15] and CodeCast [24] exploit network coding to improve performance and reliability. In all of them, the TX-RX scheme of Glossy is replaced by a sequence of indistinct slots in which a node dynamically decides whether to TX or RX; in Chaos, a TX happens deterministically when the node observes new information affecting the global aggregate, while in Mixer and CodeCast the decision includes also a probabilistic component.

Our research endeavor differs from the above on two accounts. First, it explores a strategy in which slots are not indistinct, rather they have a preassigned role, as in Glossy. However, differently from Glossy, our scheme is capable of deterministically intertwining multiple flows from multiple initiators whose dissemination and termination we govern with a novel, adaptive strategy as described in §3 and §4. Second, instead of many-to-many, we tackle convergecast traffic. This communication pattern is arguably more popular, thanks to its use in monitoring and data collection applications, yet hitherto dominated by systems relying on monolithic Glossy floods. By "breaking" this unit of communication and achieving better performance via individual CTX we exemplify the power of this alternate design mindset, possibly paving the way to exploration of alternate schemes catering for this and other traffic patterns.

Finally, to facilitate this exploration by others, and simplify our own system development, we follow a recent trend [15, 19, 25] and sharply separate the fine-grained CTX engine from the Weaver protocol built atop it, while taking advantage of features provided by modern transceivers that greatly simplify programming.

2.2 CTX in IEEE 802.15.4 Ultra-wideband

The remarkable performance achieved by CTX in IEEE 802.15.4 narrowband led researchers to study whether and how the same concept applies to other radios, e.g., Bluetooth Low Energy (BLE) [26] and IEEE 802.15.4 UWB [29]. The answer is not for granted, given the significantly different characteristics of the PHY layers. However, these studies have shown that both same-packet and different-packet CTX can be exploited, and their benefits still stand.

Why UWB? In this paper we focus on UWB, and specifically on the Decawave DW1000, arguably the most popular and widely-available UWB transceiver today. The DW1000, compliant with IEEE 802.15.4, is an impulse radio encoding information via short (~ 1 ns) pulses and without a carrier signal, enabling decimeter-level accuracy in distance estimation and a data rate up to 6.8 Mbps [7].

We focus on UWB for the following reasons. First, UWB is increasingly popular, as it offers both accurate distance estimation and communication. Although UWB is not as pervasive as other radios, its inclusion in smartphones by Apple and Samsung hints that this may change soon. Second, a staple UWB communication stack is currently missing, in contrast with the many available for IEEE 802.15.4 narrowband after almost two decades of work on wireless sensor networks. By focusing on convergecast, a staple feature of the latter, and leveraging the superior performance of CTX, we aim at concretely showing the effectiveness and applicability of UWB to sensing and communication scenarios, accelerating its adoption beyond localization-centric ones.

Glossy on UWB. A third motivation is that Glossy-based baselines already exist. The work in [20] reports the use of Glossy on UWB for network-wide coordination in a localization system. However, as communication is not the main focus, implementation details are scarce and no performance evaluation is provided. Recent work [23] details instead several design opportunities enabled by the DW1000 and evaluates Glossy on UWB in a testbed. Moreover, it also shows how Crystal, a representative higher-level protocol supporting convergecast (§2.1), can be "ported" to UWB with minimal modifications, yielding remarkable performance akin to the state-of-the-art one observed for narrowband [17]. Hereafter, we use the publicly-available UWB implementation of Crystal [23] as the baseline we compare Weaver against.

These systems are all enabled by nearly-simultaneous packet transmissions, similar to narrowband albeit with slightly different temporal and environmental constraints. For an in-depth analysis of the latter, and a comparison with narrowband, the reader is referred to [29]. Nevertheless, we reassert that the main contribution of this paper lies in the novel idea of merging multiple packets flows in a single network-wide flood—a notion that is not tied to UWB and therefore applicable to other radios, as further elaborated in §7.

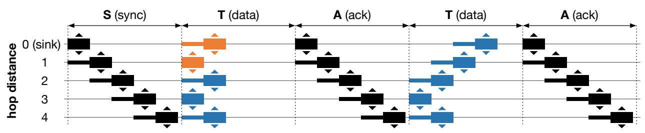
3 DESIGN GOALS AND PRINCIPLES

We designed Weaver to tackle inherent inefficiencies of data collection protocols that rely on Glossy floods as the only communication primitive. To better understand the crux of the matter, we focus on Crystal and analyze critically its operation, in particular its reliance on Glossy and the related shortcomings (§3.1). Motivated by this analysis, we then provide a concise overview of the key design decisions and goals at the core of Weaver (§3.2). We conclude the section by providing a quantitative argument, supported by analytical models, showing that our fine-grained CTX-based design is intrinsically superior to Glossy-based ones (§3.3).

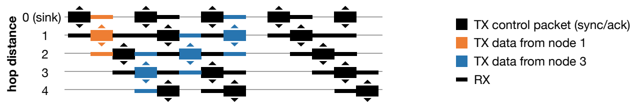
3.1 The Drawbacks of a Glossy Legacy

Crystal in a Nutshell. Crystal [17] targets scenarios with aperiodic data collection and sparse traffic, using a schedule (Figure 1a) composed of three phases, each executing a Glossy flood: *i)* the initial S phase, originating at the sink, provides network-wide time synchronization; *ii)* in the T phase, concurrent floods originating at nodes with data packets compete; due to the capture effect, one is received at the sink with high probability, e.g., the orange one in Figure 1a; *iii)* the A phase originates at the sink, which exploits it as a network-wide acknowledgment informing senders of whether their packet has been received; in Figure 1a, this enables retransmission of the blue packet in the next T phase. The alternation of T and A phases (TA pair in Crystal jargon) continues until all pending packets are received and acknowledged, and a TA pair without data is observed for a pre-defined number of times.

Crystal: A critical look. Although Crystal already achieves remarkable, state-of-the-art performance, it is inherently limited by its direct reliance on Glossy, as many others (§2).



(a) Crystal relies on distinct floods for data and control.



(b) Weaver weaves data and control into the same flood.

Figure 1: Sample executions of Crystal and Weaver. Termination phase is not shown.

A first problem is that *each Crystal phase is a Glossy flood that must complete before a new one is started.* The schedule on each node must allocate enough time for flood propagation, determined with knowledge of the network diameter and number of retransmissions.

This problem is exacerbated by the fact *that the number N of retransmissions is fixed*, yielding other inefficiencies: *i)* retransmissions are performed regardless of whether a packet has already successfully propagated, therefore hampering latency, lifetime, and possibly reliability due to unnecessary contention, and *ii)* the fixed value of N cannot dynamically cater for transient sources of unreliability, common in wireless communications; either the worst case is assumed, hampering lifetime in the normal case, or the latter is assumed, hampering reliability when disturbances occur.

These problems are shared by all Glossy-based approaches (§2). Specific to Crystal is the sink-initiated, network-wide acknowledgment in the A phase. The latter has been shown (in [17, 18] and EWSN competitions) crucial to achieve near-perfect reliability even with aperiodic, bursty traffic and heavy interference. This superior reliability motivates our use of Crystal as a baseline instead of, e.g., LWB or derivatives, besides the lack of UWB implementations. However, the asset brought by A phases also bears drawbacks, again inherited from Glossy. One directly descends from the problems above: successful propagation of a packet requires (at least) two phases, transmission (T) and acknowledgment (A), both fixed-length and strictly separated, wasting energy and time.

A less obvious problem is that *the A phase is oblivious of the reason why packets are not received.* In Crystal, the common case is that transmissions from U initiators compete in the same T phase; the A phase is crucial to inform senders of whether their packet should be re-sent. Nevertheless, the A phase also counters packet losses due to collisions. These are often concentrated in network "pockets" where packet transmissions violate the constraints for successful CTX, yielding a collision. This situation stems from a combination of neighbor density, relative signal power, and environmental conditions, extremely hard to predict yet likely to repeat due to the periodic operation of the protocol. Unfortunately, in Crystal the only option is to inform the network about the missed packet, and hope that somehow the problem solves itself.

3.2 Weaver: The Power of Fine-grained CTX

We tackle the limitations above at their core by removing the dependency on Glossy, therefore regaining the full degrees of freedom available once the unit of communication becomes an individual CTX rather than a monolithic Glossy flood. This finer-grained design mindset enables us to bring to the table several techniques that, together, improve significantly the already remarkable performance of Crystal, in its role of representative of Glossy-based approaches. The most significant point of departure is that Weaver collects and acknowledges multiple packets *within a single flood*, where the different flows coexist without disrupting each other.

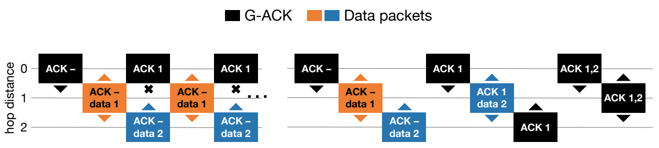
On the surface, Weaver resembles other CTX-based convergecast protocols, e.g., Crystal or LWB. Time is divided in rounds (*epochs*) of fixed length, each containing a time-slotted communication schedule. The *sink*, i.e., the node collecting data, periodically starts a new epoch by broadcasting a synchronization packet; nodes re-propagate it concurrently, exploiting CTX, and align their slots to the one of the sink, beginning execution of the global schedule.

Epoch bootstrap: Acquiring one-shot topology information. Differences begin with this first step, which in Weaver is exploited not only to enable nodes to time-synchronize, but also to *learn their hop distance from the sink.* This topology information is exploited by a node to relay only packets from nodes at the same hop distance, or higher, from the sink, i.e., favoring packets in need to make progress and quenching those already ahead, reducing contention.

This *topology information is not explicitly maintained*, as in conventional route-based approaches, *rather passively learned* during the initial dissemination, hereafter termed as *epoch bootstrap.* In this respect, the ultra-fast dissemination achieved by Weaver doubles as a fundamental asset for its operation. As we show later (§6), Weaver disseminates 30 flows over 6 hops in only ~100 ms. Therefore, *during this very short time span the network can be effectively considered as static*, and the topology learned during bootstrap safely assumed to persist throughout the entire flood, even in scenarios with node and/or sink mobility, as we investigate in §6.4.

Weaving packet flows. Weaver merges flows from multiple senders (*initiators*) into a single flood where data *implicitly* follows the upward gradient towards the sink established by the epoch bootstrap, and acknowledgments flow downwards towards initiators.

This goal is intrinsically at odds with the classic Glossy schedule alternating a TX slot with a RX one, which causes floods two hops apart to *systematically* compete. A node in RX mode hears TX from both nodes on the next and previous hop towards the sink, with the latter potentially halting propagation of data upstream. For instance, in Figure 2a, the TX of the ACK from the sink for the orange packet is performed concurrently with the TX of the blue packet. If the latter prevails, the ACK is lost and the orange packet must be retransmitted, as shown. Otherwise, the ACK prevails and delays the TX of the blue packet. Which one occurs depends on the vagaries of CTX and is therefore unpredictable, ultimately making it impossible to distinguish between data and acknowledgments.



(a) TX-RX schedule (Glossy). (b) TX-RX-RX schedule (Weaver).

Figure 2: Weaving flows of data and acknowledgments.

Weaver replaces the 2-slot TX-RX structure of Glossy with a 3-slot TX-RX-RX one (Figure 2b). This simple addition, combined with information gathered during the epoch bootstrap, decouples the TX from nodes at different hop distances, enabling each node to consistently receive *i)* in the first RX slot, data flowing upwards, i.e., from nodes *farther* from the sink and to be forwarded towards it, and *ii)* in the second RX slot, acknowledgments flowing downwards from nodes *closer* to the sink, to be forwarded to initiators.

The effect is clearly visible in Figure 2b: *CTX from nodes at different hop distances no longer interfere.* During the first RX slot, a node at hop h can receive only from senders at $h+1$, nodes at $h+3$ from senders at $h+4$, and so on. Receivers then relay data concurrently in their next TX slot, providing forwarding progress. Therefore, different data flows coexists within a single flood and proceed towards the sink in an orderly fashion. If these non-overlapping flows carry packets from different initiators, collection speed increases dramatically, as shown by comparing Figure 1b with Figure 1a.

The role of (different) acknowledgments. An obstacle remains on the path to sink: packets from same-hop initiators *compete* towards the next hop. The use of CTX ensures that one is decoded at the receiver with high probability, but what happens to the others?

Weaver solves the problem with two types of ACKs, both piggybacked on data packets whenever possible. A *local acknowledgment* (*L*-ACK) is sent by a receiver to its 1-hop neighbors to (temporarily) suppress their TX for a packet that has already made progress upwards, therefore leaving room for the propagation of other packets still behind. A *global acknowledgment* (*G*-ACK) is instead sent by the sink upon receiving a packet, and re-propagated by each node, informing the whole network that retransmissions are no longer needed for this packet, and it can be discarded.

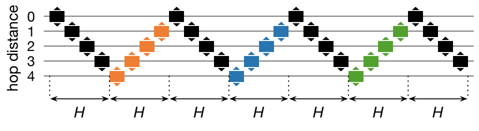
The two ACKs are implicitly related. A node whose TX is suppressed by a *L*-ACK should eventually receive a *G*-ACK; otherwise, the packet has been lost on the way to the sink, and its TX should be resumed. The crucial question then becomes: How long should the node wait before resuming TX? The answer comes, again, from the topology knowledge accrued during the epoch bootstrap that, by informing the node of its hop distance from the sink, enables an accurate estimation of the number of slots expected to elapse between a *L*-ACK and the corresponding *G*-ACK for the same packet.

3.3 Is It Worth? An Analytical Model

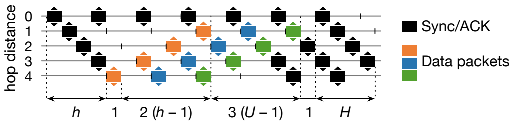
A full understanding of Weaver entails several details (§4) whose treatment we postpone to first offer evidence that our strategy achieves significant improvements w.r.t. the state of the art.

We achieve this goal with simplified models for Crystal and Weaver, where we assume that *i)* all data packets originate from U initiators placed at the same hop distance h, whose value $h = H$ is the worst-case maximum distance from the sink *ii)* packet collisions never occur, i.e., one of the packets concurrently transmitted is *always* received, and *iii)* we do not consider the overhead induced by protocol termination, present in both protocols.

Our models compute the number of slots required to disseminate U data packets under these assumptions, offering a proxy for latency and energy consumption. This allows us to *directly* compare Crystal and Weaver in an abstract setting, eliciting their *intrinsic* differences, independent of the PHY layer or other system factors.



(a) Crystal



(b) Weaver

Figure 3: Latency models for Crystal and Weaver with $U = 3$ initiators (orange, blue, and green) all at maximum hop distance H. Termination phase is not shown.

Assuming the most energy-efficient (but least reliable) configuration with a single Glossy retransmission ($N = 1$), Crystal requires

$$L_C = H(2U + 1) \tag{1}$$

CTX slots to deliver and acknowledge all U packets. H slots are required for the initial, synchronisation phase (S), followed by one TA pair with $2H$ slots for each of the U initiators.

This is exemplified in Figure 3, which also shows how Weaver significantly increases the parallelism of the U data flows and their ACKs. In the worst-case scenario we consider, the nodes $h = H$ hops away from the sink must wait h slots before they can TX data, to first receive the epoch bootstrap packet from the sink. On the other hand, differently from Crystal, TX can begin immediately after, as in Weaver CTX occur free from the many constraints of Glossy floods. The first packet reaches the sink after $1 + 2(h - 1)$ slots, as it takes 2 slots to relay a packet one hop upwards. The remaining $U - 1$ packets reach the sink once every 3 slots, completing after $3(U - 1)$ slots. Finally, the network-wide *G*-ACKs triggered by the sink upon receipt of each packet account for $1 + H$ slots each, yielding

$$L_W = 3(h + U - 1) + H \tag{2}$$

as the total number of slots utilized by Weaver.

Figure 4 compares the protocol performance based on our simplified models, for several values of network diameter H and initiators U. Crystal is more efficient in the (degenerate) case of 1-hop networks, as it uses a 2-slot schedule instead of the 3-slot one used by Weaver and, for the same reason, latency is marginally better with a single initiator ($U = 1$) at $h = H$. However, in a multi-hop network, the number of slots required by Crystal is directly proportional to the network diameter H. This is not the case for Weaver, which is also faster and more scalable as U increases due to its ability to

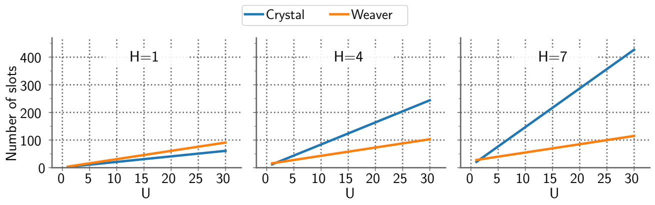


Figure 4: Estimated number of slots required to deliver U data packets in a network of H hops.

parallelize flows, up to ~2x faster for a 4-hop diameter and ~4x for a 7-hop one with $U = 30$ (Figure 4).

Although the magnitude of the performance gap between the protocols is evident, and sufficient to confirm the validity of our design choices, there are obviously several aspects that are not captured by our simplistic models. Specifically, they do not cater for system and environmental factors affecting reliability and, in turn, latency and energy consumption. These can be ascertained only with real-world experiments, which we present in §6 after further detailing our protocol and its implementation.

4 PROTOCOL DETAILS

We now complete the description of the Weaver protocol with additional, important details.

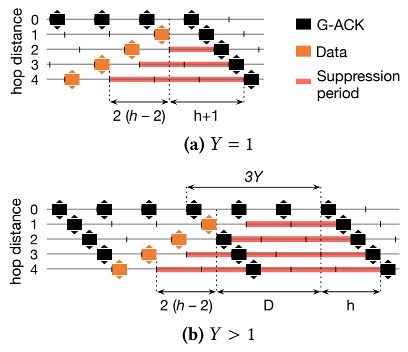
Epoch bootstrap. The bootstrap packet sent by the sink and re-propagated throughout the network at the beginning of each epoch is key to provide nodes with a common time reference and topology information. Initiators can *immediately* transmit data inside the re-propagated bootstrap packet; unlike Glossy-based systems, there is no need to separate data collection from the initial synchronization.

In theory, one bootstrap network-wide flood is enough; in practice, it may not reach all nodes due to collisions. When this happens, functionality is impaired; nodes that missed the bootstrap packet are unaware of their hop distance from the sink and do not know how to realign their schedule, preventing reliable operation. This is less of a problem with several initiators, as the hop distance included in all packets gives nodes that missed the bootstrap multiple chances to realign; however, it is crucial with few initiators.

The nodes that missed the bootstrap packet can reuse the information learned during the previous epoch. Often, this information is unchanged and can be refreshed in the next epoch, if collisions are rare. However, this may be not enough to accommodate the vagaries of wireless communication, or scenarios encompassing mobility. A simple and more reliable solution is to retransmit the bootstrap packet a pre-defined number B of times. We analyze the impact of the value of B on reliability and energy efficiency in §6.2.

Local acknowledgments (*L*-ACKs). Upon packet TX, nodes embed the initiator ID of their last heard packet in the Weaver header. When received at another node in the second RX slot, the one devoted to communication from upstream nodes, this ID indicates that the corresponding packet has already made progress towards the sink. Therefore, the original data packet doubles as a *L*-ACK for previous packets at downward nodes waiting to TX the same old data; these nodes can suppress this unnecessary packet TX and replace it with one for a new packet, if any, speeding up propagation of the latter and avoiding unnecessary contention due to the former. Nodes without new data listen during TX slots to hear same-hop neighbors and help them deliver their packets.

Global acknowledgments (*G*-ACKs). The *G*-ACKs sent by the sink contain a bitmap with one bit for each node in the system, signaling whether a packet from the corresponding node has been delivered at the sink during the epoch. *G*-ACKs are interwoven with data collection; they are received in the second RX slot from upward nodes and subsequently propagated downwards in the next TX slot. As with *L*-ACKs, nodes piggyback the *G*-ACK bitmap on data packets, if any, as part of the mandatory Weaver header.



(a) $Y = 1$

(b) $Y > 1$

Figure 5: Determining the suppression period L.

Nodes without data to send re-propagate the *G*-ACK as a no-payload packet only if it contains new bits, to reduce contention.

Linking the two ACKs: Suppression period L. The reception of a *L*-ACKs by a downward node suppresses the TX of the corresponding packet. Nevertheless, the latter must eventually be acknowledged by the sink via a *G*-ACK; if this does not happen, the packet never reached the sink and dissemination must be resumed. This combination of acknowledgments exploits spatial diversity and, as we verified experimentally, is more reliable than triggering retransmissions only upon a missed *L*-ACK, which is prone to packets remaining stuck in areas with weak links towards parents.

The availability of topology information passively gathered during epoch bootstrap enables an accurate estimation of the number L of slots expected to elapse between the RX of an *L*-ACK for a packet and the *G*-ACK (Figure 5a). Indeed, for a *G*-ACK to be sent, the corresponding packet must first be received; for an initiator at h hops from the sink, this requires $2(h-2)$ slots after RX of the *L*-ACK. At the sink, because of the 3-slot scheme, an additional slot elapses between packet reception, in the first RX slot, and the next TX. Finally, in the latter slot the sink disseminates the *G*-ACK, which travels back to the initiator, requiring additional h slots. Therefore, upon receiving a *L*-ACK, a node computes a *suppression period*

$$L = 2(h-2) + h + 1. \qquad (3)$$

If the suppressed packet is not acknowledged by the sink after L slots, the node resumes its transmission.

Both types of ACK are not immune from packet loss due to collisions, potentially causing wasteful TX that nonetheless rarely affect reliability. A missed *L*-ACK prevents packet suppression. As for *G*-ACKs, when a node receives a packet already known to be acknowledged by the sink, it resumes the piggybacking of the *G*-ACK bitmap, to cater for nodes that may have missed it.

In summary, *i)* *L*-ACKs avoid wasteful retransmissions of packets, hampering the progress of others *ii)* *G*-ACKs achieve the same goal definitely and globally *iii)* together, as determined by L, they avoid that a packet stuck in a "dead end" area is lost and forgotten.

Tuning Weaver: Batching *G*-ACKs. Acknowledgments bring several benefits, but also cause their share of problems.

Consider the example in Figure 6. Nodes B and C are at the same hop distance from the sink; their schedule is aligned and their packets, whether containing data or ACKs, compete in the same TX slot. A problem arises if one node enjoys better link quality

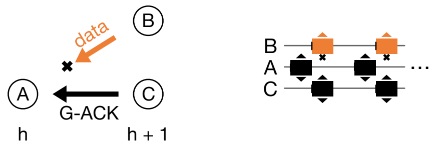


Figure 6: Example scenario in which *G*-ACKs block the propagation of data packets.

than the other(s) towards their upward node A, e.g., because C is physically closer to A. In this case, the G-ACKs re-broadcast by C are likely to suppress the data TX from B at A, and do so repeatedly due to the periodicity of schedules, until G-ACK propagation ends.

To counter situations like this, which do occur in practice, we introduce a *batching period* Y for G-ACKs at nodes other than the sink. Instead of immediately re-propagating a G-ACK upon RX, nodes send a cumulative G-ACK once every Y executions of the 3-slot TX-RX-RX pattern, i.e., every $3Y$ slots (Figure 5b). When the TX of a G-ACK occurs on a node, its bitmap is up-to-date w.r.t. G-ACKs received during this period. Therefore, the same information is delivered to the network, but $3(Y-1)$ slots are now free from data/ACK interference like the one in Figure 6. However, if data packets are transmitted in the meanwhile, the G-ACK bitmap is still piggybacked on them, as this does not cause problems.

As this technique changes the mechanics of G-ACK propagation, we revisit the earlier definition (3) of the suppression period L as

$$L = 2(h-2) + h + D \tag{4}$$

accounting for the additional D slots (Figure 5b) introduced by G-ACKs batching. As G-ACKs are issued with a predefined, globally-known period, nodes can autonomously determine the value of D upon receiving a L-ACK for a packet and before its next G-ACK.

Termination. Weaver targets fast, reliable, and energy-efficient data collection. This entails quickly turning off the network upon detecting absence of data packets while ensuring that key nodes do not leave before all packets have been delivered to the sink.

Every node terminates and enters low-power mode (sleep) after accumulating in a termination counter T a given number of inactive slots in which no new data is received. RX errors are considered an attempt from neighbors to transmit new information, and reset T. Similarly, the RX of G-ACKs informs a node that the sink is still active and whether it is aware of the data the node already transmitted; if it is not, the node postpones its termination.

The value of T depends on the protocol phase. During the epoch bootstrap, $T = 3H + 3B$, where B is the number of bootstrap packets sent and H the maximum hop distance of nodes from the sink. Indeed, *i)* H slots are required for the bootstrap packet to reach the farthest possible initiator and enable its packet TX; *ii)* $2H$ slots are required, due to the 3-slot scheme, for a packet from this worst-case initiator to reach the sink; however, *iii)* this packet competes with the B bootstrap packets rebroadcast by neighbors in consecutive TX slots; therefore, in the worst case where these are always received upstream instead of the packet, additional $3B$ slots must elapse.

If the sink does not receive any data packet T slots after sending the bootstrap packet, it enters sleep. Otherwise, an alternate counter is defined and reset every time a data packet is received. The sink waits $T = 3H + 3$ slots at the end of every G-ACK batching period. Similar to the above, *i)* H slots are required for the G-ACK to propagate downwards and enable the TX of a new packet, *ii)* $2H$ slots are required to collect it at the sink, plus *iii)* 3 slots to account for the worst case where the re-propagation of the G-ACK by a same-distance neighbor blocks the data TX. The suppression period L after a L-ACKs was defined precisely to allow transmissions to resume timely for packets that did not receive the G-ACK, giving them another chance to reach the sink before termination.

Once the sink decides to terminate, it floods a special packet to shutdown the entire network before entering sleep; there is no point in keeping nodes awake if the sink is not. However, nodes are also capable of entering sleep autonomously, as they maintain the same termination counter T as the sink; this serves as a fallback ensuring node termination when the shutdown packet is lost.

5 A MODULAR IMPLEMENTATION

Weaver relies on the ability to individually manage CTX. This, however, involves low-level radio programming, time slot management and synchronization, i.e., tedious and repetitive work that complicates and distracts from the high-level protocol logic and, worse, must be largely modified when the latter changes.

To simplify our iterative development, and enable other researchers to build their own protocols atop fine-grained CTX (§7), we designed our prototype to sharply separate these two layers. We implement the low-level functionality necessary to CTX in generic and reusable way, available to protocol designers via a simple yet expressive API (§5.1): the Time Slot Manager (TSM). The actual Weaver protocol is implemented as a thin veneer atop it, easily replaceable and modifiable. The architecture of our prototype (Figure 7), implemented on Contiki [9], is completed by an optional module enabling accurate estimation of energy consumption (§5.2).

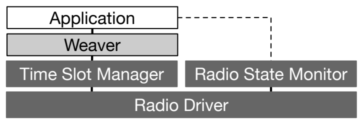


Figure 7: System architecture.

5.1 Time Slot Manager: A Flexible CTX Engine

Our goal is to avoid complexity when implementing simple things while giving fine-grained control to the higher layer, when needed.

Enabling factors. The decoupling of protocol logic is enabled by new capabilities of modern radio chips, allowing access to internal high-precision timers for timestamping radio RX events and scheduling TX/RX operations at specified times.

Without these capabilities, meeting the strict timing requirements of CTX forced protocols to trigger the next action right within the handler of the previous radio event and keep the duration of event processing constant for all nodes, packets, and protocol states to guarantee that all nodes trigger the next operation at the same time. This approach limited code branching and therefore the complexity of the higher-level protocol logic. Instead, if the next operation is scheduled directly via the internal timer of the radio, autonomously from the MCU, the protocol logic can become more rich and dynamic, expanding through levels of abstraction and indirection. The only requirement is that event processing finishes within the predefined deadline, leaving enough time for the radio to initialize and perform the next operation at the scheduled time.

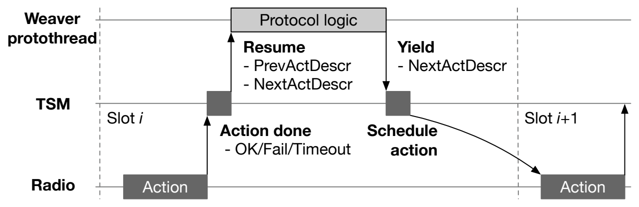


Figure 8: Flow of control.

Basic principles. We observe that all CTX systems (§2) share the same time structure. They organise communication in rounds (epochs), placing the radio to sleep between them. Each round consists of multiple fixed-duration time slots associated with a TX or RX operation (sometimes neither) and related data processing; the number of slots per round may vary. The transmitted messages contain the current slot index within the round, enabling receivers to establish the round reference time (i.e., its beginning) from the start-of-frame delimiter (SFD) timestamp of the received packet.

We delegate to TSM all this common bookkeeping related to synchronization, i.e., computing the round reference time, executing radio operations at the right times, and updating the synchronization information in the header of TX packets to allow reference time to propagate over multiple hops. Instead, we leave it to the higher layer (e.g., WEAVER) to decide what action (TX, RX, or none) to perform in each slot, the data payload for each TX slot, and when to stop the current round and enter sleep until the next begins.

A node joins the network via the TSM_SCAN operation, instructing TSM to listen to the channel until a packet is received. When this happens, TSM automatically synchronizes with the network and starts the slotted operation. Optionally, the higher layer can instruct TSM to adjust its reference time upon any successful RX; it is wise to do so periodically, to counter clock drift. Unlike Glossy, TSM is agnostic of node roles, leaving it to the higher layer to determine which node(s) provide the authoritative time reference, but provides all the necessary time calculations, adjustments, and scheduling.

API and control flow. A protocol built atop TSM begins with a TSM_START call providing the desired slot duration and a pointer to the slot handler function. The control flow is then driven by TSM, which automatically calls the latter function before every slot (Figure 8), passing as a parameter a special read-only structure describing the operation performed in the *previous* slot, if any, including the code of the action performed, its status (success or error code), the RX payload data and size (if any), and the slot index.

Another structure describing the *next* slot action is passed by reference, to be filled by the slot handler function. TSM pre-configures most fields with default values based on settings and context; only few must be set by the protocol. This next-slot structure includes the action to perform (SCAN, RX, TX, RESTART, STOP), a pointer to the TX payload or RX buffer, and other fields described later. After the slot handler function ends, control returns to TSM which uses the values set in this structure to schedule the next action.

In principle, a conventional function can be used as slot handler. However, TSM was designed to take full advantage of Contiki protothreads by providing convenience calls (C macros) that combine the configuration of the next action with protothread interaction, effectively mimicking a conventional blocking function (Figure 8). These convenience calls yield control to the system, letting the MCU perform other tasks or go to a low-power mode while waiting for the requested action to complete; when this occurs, the protothread is resumed from the point where it requested the TSM action. This enables the description of a complex protocol logic as a sequential program with branching and loops, arguably more natural than the cumbersome event-driven style necessary with classic callbacks.

```
while(1) {
  TSM_SCAN(buf); // scan until RX
  if (prevActDescr.status == SUCCESS) {
    // forward the packet in the next slot
    TSM_TX(buf, prevActDescr.data_len);
    TSM_RESET(ROUND_PERIOD); // skip to the next round
}}
```

Figure 9: Glossy forwarder logic ($N = 1$) atop TSM.

Figure 9 shows a naïve yet working implementation of the Glossy forwarder logic with $N = 1$, written atop TSM in only 6 lines of code. A full-blown Glossy re-implementation is outside the scope of this paper; the code is meant to illustrate the simplicity induced by TSM, fully exploited in our WEAVER prototype. Each loop iteration is a Glossy round. The forwarder, i.e., any node other than the initiator, begins each round by scanning the channel for incoming packets. When one arrives, TSM automatically uses it to resume the protothread and begin slotted operation; in case of successful RX the node retransmits the exact same payload in the next slot via the TSM_TX call. TSM advances the slot index automatically and updates the TX packet header accordingly. After packet TX is finished, the TSM_RESET call instructs TSM to finalize the current round and sleep for the rest of the specified ROUND_PERIOD.

This example shows how TSM keeps simple things simple, by hiding all operations related to timing and slot scheduling, and allowing the higher layer to concentrate on the protocol logic. On the other hand, the fact that WEAVER, a significantly more complex protocol, was implemented atop TSM confirms that the abstractions in TSM are not only simple but also expressive.

Delayed TX. As reported in [29], CTX perform significantly better in UWB if they are slightly de-synchronized, unlike in narrowband. TSM caters for this by allowing the definition of a small TX delay (ns to μs) on a per-slot basis. This requires adding the value of the delay used to the nominal slot reference in the TSM header, enabling receiving devices to compensate the delay when computing the round time reference. WEAVER exploits this feature by inserting a random delay before all TX, specified in the corresponding field of the next-action structure. Delay values are reported in §6.1.

RX timeouts and energy savings. Idle listening (preamble hunt) is the most energy-consuming operation of the DW1000. Therefore, we minimize the time the radio listens to the channel in RX slots. Since we expect nodes to be synchronized, the radio can begin listening shortly before we expect the frame preamble to arrive, and stop shortly after if none is received. We achieve this by setting

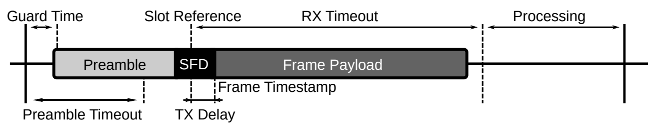


Figure 10: TSM slot structure.

a preamble detection timeout equal to the sum of *i)* the initial guard time *ii)* the maximum TX delay senders could use, and *iii)* half the preamble duration. If no preamble symbols are detected before timeout, the radio switches to idle mode automatically and triggers an event to TSM. Moreover, we set a frame RX timeout to guarantee that any RX operation leaves enough time within the slot for the protocol layer to run its logic and prepare for the next slot. Figure 10 shows the resulting structure of a RX slot. In case a timeout triggers, TSM reports a failed RX action to the higher layer.

5.2 Monitoring Energy Consumption

An accurate estimation of energy consumption is crucial to validate the performance of our prototype. Systems built atop Contiki for IEEE 802.15.4 narrowband can rely on the well-known Energest [10] component. Unfortunately, no equivalent exists for UWB.

Therefore, we designed our own component to estimate the energy spent during radio operations. Our Radio State Monitor module (Figure 7) brings the core concepts of Energest to the more complex state machine of the DW1000 radio. This entails supporting several key features not present in Energest, e.g., delayed operations and timeouts, and using the precise timer of the radio. As in Energest, our module maintains several counters aggregating the overall time spent by the radio in the various states. However, differently from it, our module tracks separately different portions of the frame RX and TX for more accurate estimation of the energy spent, as these consume very different amounts of energy on the DW1000. Finally, the current drawn in idle mode is also accounted for.

Overall, the Radio State Monitor is a valuable contribution per se, not tied to CTX, that can be exploited by other researchers working on UWB at large to assess the energy consumption of their systems.

6 EVALUATION

We evaluate WEAVER in a UWB testbed at our premises, considering two topologies with different characteristics. We first provide an in-depth analysis of parameters B and Y, which control the reliability of the epoch bootstrap and the periodicity of G-ACK dissemination, and analyse their impact on performance. We then compare WEAVER to the UWB implementation of Crystal [23] in the same conditions. For both protocols we report *i)* the packet delivery rate (*PDR*) at the sink, *ii)* the per-epoch estimated energy consumption of non-sink nodes, and *iii)* the latency, defined as the time between the beginning of an epoch and the delivery of the *last* data packet at the sink. Finally, we experiment with mobile nodes to assess whether WEAVER is suitable for use in dynamic topologies.

6.1 Experimental Setup

Hardware and testbed. We report experiments from a 36-node testbed installed on the ceiling above the corridors of an office building, over a 84×33 m^2 area (Figure 11). Each node includes a Raspberry Pi, a JTAG programmer, and a DecaWave EVB1000 board equipped with a DW1000 UWB radio and a STM32F105 ARM Cortex M3 MCU. A dedicated Ethernet infrastructure enables automated and remote control of experiments and collection of logs.

Network topologies. We consider two topologies, called FLOOR and LINEAR, with different characteristics. In FLOOR, node 1 is designated as the sink, all nodes are active, and the network spans 3 hops.

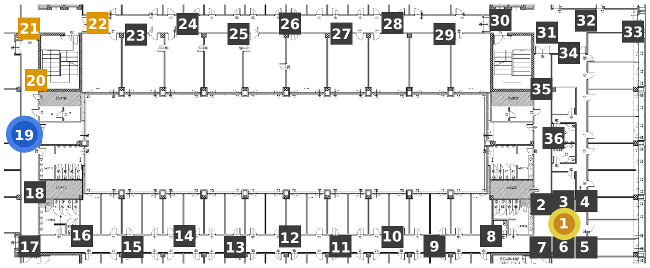


Figure 11: Testbed spanning $84 \times 33m^2$. In FLOOR, node 1 is the sink. LINEAR excludes node 20–22; node 19 is the sink.

Data can flow along two paths—clockwise and counter-clockwise—providing spatial diversity. The sink is deployed in a dense area where 10 neighbors have near-perfect link quality towards it, and most (nodes 2–7) are placed at similar distances from it. As reported in [29], a similar scenario can be challenging for CTX-based protocols, and therefore intriguing to analyze, since multiple signals with similar strengths and timing are likely to reach the sink, increasing collisions especially with several and different packets.

In LINEAR, node 19 acts as the sink and nodes 20–22 (top left corner) are disabled, preventing communication between the sink and node 23. This *i)* increases the maximum hop distance to 6 hops, and *ii)* forces all data flows to proceed along a single path, significantly reducing spatial diversity. Moreover, node 18 cannot communicate with any of nodes 8–16 on the bottom corridor; therefore, node 17 is the *only* connection between the sink and the remaining (large) part of the network. The absence of receiver redundancy, known to be detrimental for CTX-based protocols, makes this topology particularly challenging, yet realistic in indoor environments.

Radio configuration. We use channel 4 with 64 MHz pulse repetition frequency (*PRF*). To minimize energy consumption, we choose the highest 6.8 Mbps data rate on the DW1000 and the shortest ~64 μs preamble, in line with [23]. We set the TX power to the maximum recommended [8] for our channel and *PRF*. We exploit TSM to randomly delay all transmissions by up to 1 μs (i.e., roughly the duration of one preamble symbol) as this small de-synchronization significantly reduces the chance of collision in UWB [29].

Packet size and slot duration. Long packets are known to increase the chance of collision when transmitted concurrently [12, 23]. To assess how this impacts reliability and energy consumption, we perform experiments with both short (2B) and long (100B) payloads. We set the duration of WEAVER slots to 813 μs, enough to accommodate the maximum IEEE 802.15.4 frame length.

6.2 Dissecting WEAVER

We study the impact of parameters B and Y on the performance of WEAVER. The former impacts the reliability of epoch bootstrap, while the latter controls the trade-off between latency and energy consumption depending on the expected traffic patterns.

Reliability of epoch bootstrap. We explored $B \in \{1..3\}$. Table 1 reports the number of failed epoch bootstrap attempts across 10,000 epochs, with no initiators ($U = 0$). In FLOOR, $B = 1$ yields 59 occurrences of a node missing the bootstrap packet (0.015%), while $B = 2$ yields only 1 occurrence (0.0003%). LINEAR is less prone to a

Table 1: Occurrence of failed bootstrap for any node and average energy consumption vs. number B of bootstrap packet retransmissions. Data acquired over $10,000$ epochs in two topologies with no initiator ($U = 0$).

	% of failed bootstraps			Energy (mJ)		
Topology	B=1	B=2	B=3	B=1	B=2	B=3
Floor	0.015	0.0003	0	2.49	3.00	3.35
Linear	0.0016	0.0006	0	3.19	3.46	3.73

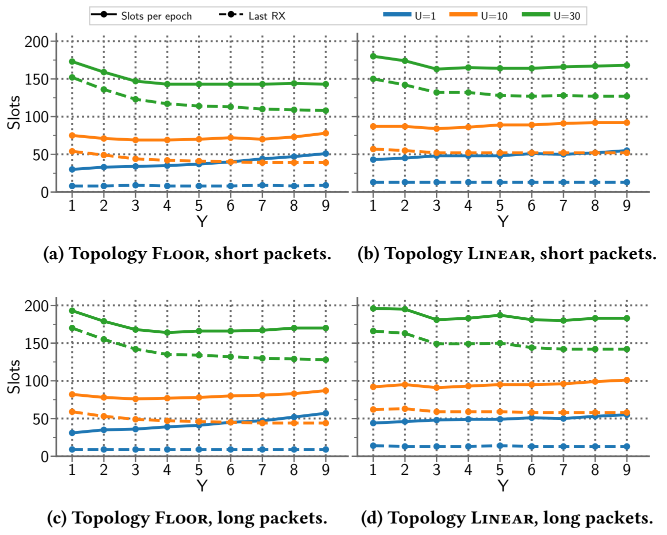
failed bootstrap, with the same values of B yielding only 5 and 2 occurrences (0.0016% and 0.0006%), respectively.

The value $B = 3$ guarantees a correct bootstrap in all epochs for both topologies. However, this reliability comes at the cost of energy consumption (Table 1) whose increase is more evident without traffic ($U = 0$) as node termination directly depends on B (§4). In this case, consumption is 3.35 mJ and 3.73 mJ in Floor and Linear, a +35% and +17% increase w.r.t. $B = 1$. However, when traffic is present ($U > 0$), the influence of B is less marked as *i)* the network remains awake for longer to collect all data, and *ii)* collection starts immediately, in parallel with bootstrap. Hereafter, we set $B = 2$, the best compromise between reliability and energy efficiency.

Impact of *G*-ACK batching period. The period Y used to disseminate *G*-ACKs upon data reaching the sink is the main knob to control Weaver, balancing timeliness in acknowledging packets via *G*-ACKs with their interference with data (§4). We analyze the impact of Y on the duration of the flood (Figure 12). For each combination of topology, packet size, $Y \in \{1..9\}$, and $U \in \{1, 10, 30\}$ the results are obtained by aggregating 1000 epochs.

The impact of Y on termination, while not high in relative terms, varies in function of the amount of traffic (Figure 12). In both topologies, Weaver shows a similar response to the increase of Y, although the trend is more evident in Floor. Similarly, packet size does not have a substantial impact, with longer packets causing only a slight increase in latency. With sparse traffic, increasing Y does not yield benefits as *G*-ACKs rarely interfere with data floods, making the duration of the collection phase independent from Y. Thus, $Y = 1$ is the fastest and the most energy-efficient solution, as it minimizes the time a node waits in between the last packet RX and termination (§4). However, as the number U of initiators increases, a small value of Y becomes detrimental; with $Y = 1$, each packet reaching the sink triggers a new *G*-ACK, disseminated network-wide. This increases contention and the risk of interference between data and *G*-ACKs, slowing down the collection process. By increasing Y and therefore reducing the number of *G*-ACKs, we increase the chance to collect multiple packets in between two consecutive *G*-ACKs floods. The impact on the flood duration is clearly visible for $U = 30$. For instance, in Floor and with short packets a flood requires 174 slots to terminate with $Y = 1$, and only 143 with $Y = 4$ (−18%). On the other hand, increasing Y further does not pay off, as it forces the system to remain active for several slots after the last packet collected; this is very costly with sparse traffic and brings little to no improvement with a denser one.

The best choice of Y ultimately depends on the behavior of initiators. If the traffic profile is known beforehand, users can tune the value of Y to further reduce the latency and energy consumption



(a) Topology Floor, short packets. (b) Topology Linear, short packets.
(c) Topology Floor, long packets. (d) Topology Linear, long packets.

Figure 12: *G*-ACK batching period Y vs. number of slots required for termination and last packet collected at sink.

of Weaver. Otherwise, Figure 12 shows that the impact is relatively limited anyway. In the rest of this section, we assume the application has no a priori knowledge of traffic and set $Y = 4$ as in our case this is a good balance across all dimensions.

Finally, Weaver achieved $PDR \geq 99.9\%$ independently from the value of Y. A more thorough study of reliability is described next.

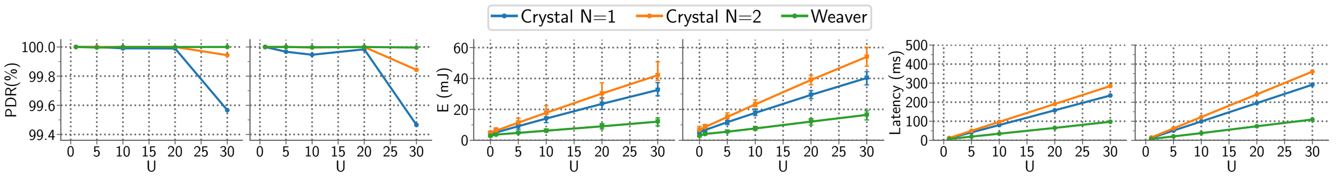
6.3 Weaver vs. Crystal

We compare against Crystal [17], a state-of-the-art data collection protocol, using its publicly-available implementation for UWB [23].

Protocol configurations. We configure Weaver with $B = 2$ bootstrap packet retransmissions and a *G*-ACK batching period $Y = 4$, informed by our analysis in §6.2. Configuring Crystal entails tuning the underlying Glossy for every phase, by defining N and adapting the maximum phase duration W accordingly. Large values of N enhance flood reliability, by increasing the spatio-temporal redundancy of Glossy, but also increase energy consumption. In our analysis we consider $N \in \{1, 2\}$, exploring different trade-offs between reliability and energy efficiency. $N = 1$ is the most energy-savvy configuration possible, but also the most fragile. Table 2 reports a summary of the configurations. Following the methodology of [17], we dimension W for each phase to accommodate the maximum hop count H, plus a small slack to cope with possible flood delays due to collisions. Other Crystal parameters (e.g., number of empty TA pairs before termination) are unchanged w.r.t. [17, 23].

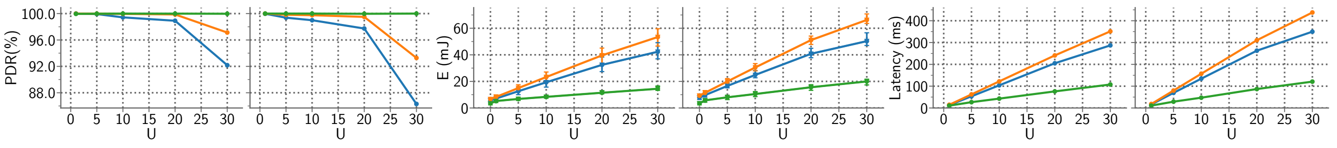
Results. For each combination of topology, number of initiators $U \in \{0, 1, 5, 10, 20, 30\}$, packet size, and protocol configuration, we collect execution traces of 5000 epochs for both protocols.

In Floor, Weaver is largely unaffected by packet size, achieving near-perfect reliability with both short and long ones (Figure 13a, 13b) even under heavy contention, with $PDR > 99.99\%$ when $U = 30$. Instead, the reliability of Crystal is significantly lower, especially with $N = 1$, and the negative impact of long packets is clearly visible as U increases, due to the higher chance of collisions.



(a) *PDR*, **short packets.** **(b)** *PDR*, **long packets.** **(c) Energy, short packets.** **(d) Energy, long packets.** **(e) Latency, short packets.** **(f) Latency, long packets.**

Figure 13: WEAVER vs. Crystal in the FLOOR topology.



(a) *PDR*, **short packets.** **(b)** *PDR*, **long packets.** **(c) Energy, short packets.** **(d) Energy, long packets.** **(e) Latency, short packets.** **(f) Latency, long packets.**

Figure 14: WEAVER vs. Crystal in the LINEAR topology.

We actually found an increased rate of collisions for long packets also in WEAVER, by analyzing the RX error rate at the level of single slots. For instance, with $U = 30$ each node incurs in a RX error 5.83 times per epoch with short packets and 9.07 with long ones. However, WEAVER can tolerate more collisions, as the continuous flood grants each node many chances to retransmit. Further, the number of retransmissions is not fixed beforehand, as in Glossy and therefore Crystal, rather it adapts to data traffic, as nodes keep attempting to forward packets upon collisions. Moreover, thanks to the L-ACKs, WEAVER can promptly suppress transmissions before the arrival of G-ACKs from the sink, quickly reducing contention.

High reliability often comes with extra energy consumption. This is not the case for WEAVER, specifically designed to remove the inefficiencies of Glossy-based solutions. Indeed, the fast, reliable and contention-resilient operation of WEAVER yields significant energy improvements w.r.t. Crystal (Figure 13c, 13d).

Without traffic ($U = 0$), WEAVER consumes 40% and 57% less than Crystal with short and long packets, respectively. The benefits of fine-grained control over CTX increase with U, as WEAVER fully unleashes its ability to parallelize collection floods, *reducing energy consumption by* ~70% *for* $U = 30$ *initiators regardless of packet size.*

In the FLOOR topology explored so far, multiple paths enable packets to reach the sink, itself surrounded by many relays. In LINEAR, all data towards the sink must flow through the bottleneck of node 17; continued collisions at this node can lead to interruption of the flood and multiple packet losses. Crystal behaves poorly in these conditions; even with the more reliable $N = 2$, *PDR* decreases as U increases, down to ~97% and ~93% for $U = 30$ and short and long packets, respectively (Figure 14a, 14b). WEAVER is affected to a much smaller extent, achieving *PDR* > 99.9% in all conditions.

Energy consumption increases for both protocols in LINEAR, due to the larger diameter. However, WEAVER consumes a fraction of the energy required by Crystal (Figure 14c, 14d) similar to what observed in FLOOR; with $U = 0$, WEAVER saves 40% and 63% with short and long packets, and ~70% for both packet sizes with $U = 30$.

WEAVER is highly reliable and energy efficient in both our topologies, despite LINEAR being quite challenging. The question is whether it is also faster, as predicted by our model (§3.3). Many definitions of latency are possible. We report the time needed to complete data collection (i.e., RX of last packet) because *i)* the average latency incurred by a packet is roughly half the duration of collection, and *ii)* flood termination at the sink happens consistently a few slots after the RX of the last packet, making these two metrics redundant.

Our experiments confirm that WEAVER is significantly faster than Crystal. With long packets and $U = 30$, Crystal receives the last packet after 361 ms in FLOOR, and 438 ms in LINEAR, while WEAVER does the same in only 109 ms and 121 ms, respectively (Figure 13f, 14f). Interestingly, switching from FLOOR to LINEAR causes a 21% latency increase for Crystal, but only 11% for WEAVER. Even with a single packet ($U = 1$) WEAVER is faster at 7 ms and 11 ms, against the 15 ms and 18 ms of Crystal, also thanks to the ability to begin packet TX concurrently with the initial bootstrap. WEAVER is faster than Crystal also with short packets, in both topologies (Figure 13e, 14e). For $U = 30$, its latency is 97 ms and 107 ms, against 285 ms and 351 ms for Crystal. For $U = 1$, WEAVER incurs a latency similar to long packets, while the one of Crystal, reduced to 12 ms and 15 ms, remains higher than WEAVER.

Optimizing the fixed slot duration (§6.1) for packet size leads to large improvements. For 2B packets, a shorter slot of 455 μs reduces latency and energy consumption respectively by 43% and 22% w.r.t. Figure 13–14, regardless of traffic and without affecting reliability.

Table 2: Parameters used for the two configurations of Crystal considered. T_s and T_l are the duration of the T phase optimized for a short (2B) and long (100B) packet, respectively.

Topology	N	W (ms)			
	(S,T,A)	S	A	T_s	T_l
FLOOR	1	2.7	2.8	2.8	4.5
	2	3.6	3.7	3.6	6.1
LINEAR	1	4.0	4.1	4.0	6.0
	2	4.8	5.0	4.9	8.4

6.4 WEAVER and mobility

Among the advantages of CTX-based protocols is that they are agnostic of the underlying network; being resilient to topology changes they are suitable for scenarios with mobile nodes [11]. However, this is not entirely true for Weaver. Nodes learn their hop distance during the epoch bootstrap, and leverage this information to direct data and *G*-ACKs flows during collection; this potentially makes the protocol susceptible to topology changes.

Nevertheless, Weaver completes the collection of packets from 30 initiators over a 6-hop network in ~100 ms (Figure 14e). During this time, a person walking covers ~14 cm and a car traveling at 100 km/h covers ~3 m. A Weaver flood is so fast that even when nodes are moving the topology inside it remains essentially static.

We ascertain whether this is true, and the applicability of Weaver to mobile scenarios, through experiments in which 3 people, each carrying a node, walk at brisk pace in the testbed area for the entire duration of the test. In these experiments, all 39 nodes of the testbed are active. As mobile nodes traverse the testbed, their links to other nodes degrade or even interrupt abruptly due to obstacles.

Table 3 shows *PDR*, latency, and energy consumption with a static or mobile sink. In the first scenario, node 1 is the sink, as in Floor, and all mobile nodes are initiators; in the second, one of them serves as mobile sink. The latter scenario is particularly challenging, as sink movement *i)* alters the structure of the collection scheme, and *ii)* explores several topologies at once, including problematic ones like Linear. We run 2000 epochs for each $U \in \{10, 30\}$ and short packets, and observed no packet loss with $U = 10$ and $PDR > 99.9\%$ with $U = 30$, regardless of sink mobility.

Overall, the values in Table 3 are in line with those in §6.3; mobility appears to have little to no impact on performance. This confirms the resilience of Weaver w.r.t. mobility, which we are evaluating more extensively as part of our future work.

7 DISCUSSION AND OUTLOOK

We concisely elaborate on the potential impact of our work and how it could be extended and generalized by other researchers.

What did we accomplish? The evaluation we presented, along with the analytical model in §3.3, confirm that protocols based on fine-grained CTX rather than monolithic Glossy floods can unlock remarkable improvements over the already impressive performance achieved by the latter. The ability to weave and consolidate multiple floods into a single, coordinated one improves on latency, but also on reliability and energy consumption—i.e., all three metrics in which CTX excel. On the other hand, the very small latency also enables a novel way to exploit topology information, allowing protocol designers to treat the network as static even when it is not, as in scenarios encompassing mobility. We argue that these design principles are a contribution per se, which goes beyond the nonetheless remarkable performance of Weaver.

What about other radios? Although we focused on UWB, we argue that our contribution is not limited to it, as neither Weaver nor TSM rely on features specific to the PHY or radio chip we used.

Table 3: Performance of Weaver with 3 mobile nodes.

Sink	*PDR*		Latency (ms)		Energy (mJ)	
	U=10	U=30	U=10	U=30	U=10	U=30
Static	100	99.993	57.72	119.51	7.58	14.07
Mobile	100	99.95	61.79	123.58	8.22	15.24

The superior performance of Weaver is intrinsically determined by its use of fine-grained CTX, as shown quantitatively in §3.3. Indeed, the efficient organization of multiple data flows in Weaver builds solely on the assumption that receivers can successfully decode, with high probability, one among different packets transmitted concurrently. As mentioned, this assumption has been shown to hold for other popular PHY layers besides UWB. Therefore, we expect the principles of Weaver, if not the exact protocol, to find direct application for these other radio technologies.

However, the extent to which our quantitative findings can be transferred to other radios is yet to be established experimentally, for which TSM provides a handy framework. We argue that it is simple to port TSM to any platform that, like DW1000, supports timestamping and scheduling of packet TX and RX precisely enough to enable non-destructive interference of TX signals. A short-term item on our research agenda is to port TSM and Weaver to a modern IEEE 802.15.4 narrowband radio supporting these features, e.g., the CC2538 for which Contiki-based implementations of Glossy already exist [16], further simplifying the transfer of our results.

What about other traffic patterns? The role of TSM, however, is not limited to simplifying the transfer of our results to other platforms. On the contrary, our main motivation for its development was to sharply separate the *general* low-level mechanics of CTX from the *specific* higher-layer protocol exploiting them.

In this respect, Weaver is only one of the possibilities, geared towards data collection. We argue that the benefits unlocked by the key insight of Weaver, i.e., its fine-grained use of individual CTX instead of monolithic floods, can be reaped for other traffic patterns similar to what happened for Glossy, whose availability as a core communication primitive was exploited in many directions.

What about ranging and localization? Our focus on UWB opens intriguing opportunities. For instance, the work in [5] has recently shown that CTX in UWB enable concurrent distance estimation (ranging) towards multiple nodes at once, inspiring several follow-up works [6, 13, 14, 30]. The concepts in Weaver, and the core building blocks in TSM, could therefore be exploited to rejoin the two perspectives of communication and localization enabled by UWB under a single framework efficiently enabling both.

8 CONCLUSIONS

CTX have been studied for about a decade, but largely within the perimeter of what enabled by the popular Glossy system. In this paper, we show that an alternate design mindset is possible; one where the protocol designer regains control over all degrees of freedom enabled by using individual CTX as building blocks, significantly finer-grained than the monolithic one offered by Glossy. We offer analytical and experimental evidence that this alternate design paradigm brings remarkable advantages in the context of convergecast, and provide publicly-available, open-source software [1] enabling researchers to explore other ways to harvest its benefits.

9 ACKNOWLEDGMENTS

This work is partially supported by the Italian government via the NG-UWB project (MIUR PRIN 2017).

REFERENCES

[1] https://github.com/d3s-trento/contiki-uwb.

[2] M. Baddeley, A. Stanoev, U. Raza, M. Sooriyabandara, and Y. Jin. Competition: Adaptive software defined scheduling of low power wireless networks. In *Proc. of EWSN*, 2019.

[3] M. Brachmann, O. Landsiedel, and S. Santini. Concurrent transmissions for communication protocols in the internet of things. In *Proc. of LCN*, 2016.

[4] T. Chang, T. Watteyne, X. Vilajosana, and P. H. Gomes. Constructive interference in 802.15.4: A tutorial. *IEEE Communications Surveys Tutorials*, 2019.

[5] P. Corbalán and G. P. Picco. Concurrent Ranging in Ultra-wideband Radios: Experimental Evidence, Challenges, and Opportunities. In *Proc. of EWSN*, 2018.

[6] P. Corbalán, G. P. Picco, and S. Palipana. Chorus: UWB Concurrent Transmissions for GPS-like Passive Localization of Countless Targets. In *Proc. of IPSN*, 2019.

[7] DecaWave Ltd. DW1000 Data Sheet, version 2.19, 2017.

[8] DecaWave Ltd. DW1000 User Manual, version 2.18, 2017.

[9] A. Dunkels, B. Gronvall, and T. Voigt. Contiki - a lightweight and flexible operating system for tiny networked sensors. In *Proc. of LCN*, 2004.

[10] A. Dunkels, F. Osterlind, N. Tsiftes, and Z. He. Software-based on-line energy estimation for sensor nodes. In *Proc. of EmNets*, 2007.

[11] F. Ferrari, M. Zimmerling, L. Mottola, and L. Thiele. Low-power Wireless Bus. In *Proc. of SenSys*, 2012.

[12] F. Ferrari, M. Zimmerling, L. Thiele, and O. Saukh. Efficient Network Flooding and Time Synchronization with Glossy. In *Proc. of IPSN*, 2011.

[13] B. Großwindhager et al. Concurrent Ranging with Ultra-Wideband Radios: From Experimental Evidence to a Practical Solution. In *Proc. of ICDCS*, 2018.

[14] B. Großwindhager et al. SnapLoc: An Ultra-Fast UWB-Based Indoor Localization System for an Unlimited Number of Tags. In *Proc. of IPSN*, 2019.

[15] C. Herrmann, F. Mager, and M. Zimmerling. Mixer: Efficient many-to-all broadcast in dynamic wireless mesh networks. In *Proc. of SenSys*, 2018.

[16] K. C. Hewage, S. Raza, and T. Voigt. Protecting glossy-based wireless networks from packet injection attacks. In *Proc. of MASS*, 2017.

[17] T. Istomin, A. L. Murphy, G. P. Picco, and U. Raza. Data Prediction + Synchronous Transmissions = Ultra-low Power Wireless Sensor Networks. In *Proc. of SenSys*, 2016.

[18] T. Istomin, M. Trobinger, A. L. Murphy, and G. P. Picco. Interference-Resilient Ultra-Low Power Aperiodic Data Collection. In *Proc. of IPSN*, 2018.

[19] R. Jacob, J. Baechli, R. D. Forno, and L. Thiele. Synchronous transmissions made easy: Design your network stack with baloo. In *Proc. of EWSN*, 2019.

[20] B. Kempke, P. Pannuto, B. Campbell, and P. Dutta. SurePoint: Exploiting Ultra Wideband Flooding and Diversity to Provide Robust, Scalable, High-Fidelity Indoor Localization. In *Proc. of SenSys*, 2016.

[21] O. Landsiedel, F. Ferrari, and M. Zimmerling. Chaos: Versatile and Efficient All-to-all Data Sharing and In-network Processing at Scale. In *Proc. of SenSys*, 2013.

[22] R. Lim, R. Da Forno, F. Sutton, and L. Thiele. Competition: Robust flooding using back-to-back synchronous transmissions with channel-hopping. In *Proc. EWSN*, 2017.

[23] D. Lobba, M. Trobinger, D. Vecchia, T. Istomin, and G. P. Picco. Concurrent transmissions for multi-hop communication on ultra-wideband radios. In *Proc. of EWSN*, 2020.

[24] M. Mohammad and M. C. Chan. Codecast: Supporting Data Driven In-Network Processing for Low-Power Wireless Sensor Networks. In *Proc. of IPSN*, 2018.

[25] B. A. Nahas, S. Duquennoy, and O. Landsiedel. Network-wide Consensus Utilizing the Capture Effect in Low-power Wireless Networks. In *Proc. of SenSys*, 2017.

[26] B. A. Nahas, S. Duquennoy, and O. Landsiedel. Concurrent Transmissions for Multi-Hop Bluetooth 5. In *Proc. of EWSN*, 2019.

[27] F. Sutton, R. Da Forno, D. Gschwend, T. Gsell, R. Lim, J. Beutel, and L. Thiele. The design of a responsive and energy-efficient event-triggered wireless sensing system. In *Proc. of EWSN*, 2017.

[28] M. Suzuki, Y. Yamashita, and H. Morikawa. Low-power, end-to-end reliable collection using glossy for wireless sensor networks. In *Proc. of VTC Spring*, 2013.

[29] D. Vecchia, P. Corbalan, T. Istomin, and G. P. Picco. Playing with Fire: Exploring Concurrent Transmissions in Ultra-wideband Radios. In *Proc. of SECON*, 2019.

[30] T. Wang, H. Zhao, and Y. Shen. An Efficient Single-Anchor Localization Method Using Ultra-Wide Bandwidth Systems. *Applied Sciences*, 2020.

[31] M. Zimmerling, L. Mottola, P. Kumar, F. Ferrari, and L. Thiele. Adaptive real-time communication for wireless cyber-physical systems. *ACM Transaction on Cyber-Physical Systems*, 2017.