Davi Resner

Performance Evaluation of the Trustful Space-Time Protocol

Florianópolis

2018

Davi Resner

Performance Evaluation of the Trustful Space-Time Protocol

Dissertatação submetida ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Santa Catarina para a obtenção do Grau de Mestre em Ciência da Computação.
Orientador: Prof. Dr. Antônio Augusto Fröhlich

Florianópolis

2018

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Resner, Davi
Performance Evaluation of the Trustful Space Time Protocol / Davi Resner ; orientador, Antônio Augusto Fröhlich, 2018.
190 p.

Dissertação (mestrado) - Universidade Federal de Santa Catarina, Centro Tecnológico, Programa de Pós Graduação em Ciência da Computação, Florianópolis, 2018.

Inclui referências.

1. Ciência da Computação. 2. Protocolos de Comunicação. 3. Internet das Coisas. 4. Design cross layer. 5. Avaliação de desempenho. I. Fröhlich, Antônio Augusto. II. Universidade Federal de Santa Catarina. Programa de Pós-Graduação em Ciência da Computação. III. Título.

Davi Resner

Performance Evaluation of the Trustful Space-Time Protocol

Esta dissertação foi julgada adequada para obtenção do título de mestre e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação.

Florianópolis, 02 de março de 2018.

____
Prof. José Luís Almada Güntzel, Dr.
Coordenador do Programa

Banca Examinadora:

____
Prof. Antônio Augusto Fröhlich, Dr.
Universidade Federal de Santa Catarina
Orientador

____
Prof. Wolfgang Schröder-Preikschat, Dr.
Friedrich-Alexander-Universität

____
Prof. Antonio Alfredo Ferreira Loureiro, Dr.
Universidade Federal de Minas Gerais

____
Prof. Gustavo Medeiros de Araujo, Dr.
Universidade Federal de Santa Catarina

____
Prof. Paulo José de Freitas Filho, Dr.
Universidade Federal de Santa Catarina

À Julia.

Abstract

Wireless Sensor Networks (WSN) have been implemented in many different forms over the years. Buildings, homes, farms, rivers, the weather, assembly lines, and many more physical environments, can all be monitored and sometimes controlled by a wireless network of cheap computing devices equipped with different sensors and actuators. As these networks get connected to the Internet of Things (IoT), it is ever more important that they operate trustfully, with carefully-designed and -implemented domain-oriented operating systems and network protocols.

The Trustful Space-Time Protocol (TSTP) is a cross-layer WSN protocol designed to deliver authenticated, encrypted, timed, and georeferenced messages containing data compliant with the International System of Units (SI) in a resource-efficient way. By integrating shared data from multiple networking services into a single communication infrastructure, TSTP is able to eliminate replication of information across services, achieving small overhead in terms of control messages. However, the complexity of TSTP's features, its broad range - from application to Medium Access Control, - and its experimental nature bring diverse requirements beyond those usually considered in most software designs.

In this dissertation, the protocol design of TSTP is presented in detail, with a description of the message formats and algorithms used for medium access control, geographic routing, spatial localization, time synchronization, and security. Then, an implementation is developed for the Embedded Parallel Operating System (EPOS) and the EPOSMote III WSN platform. To avoid a monolithic implementation of the cross-layer approach, a component-based design is used, exploring template metaprogramming techniques to adapt and combine basic building blocks. An event-driven architecture that makes use of zero-copy buffers and metadata is used to handle crosscutting concerns. The design and implementation are assessed with experiments on the EPOSMote III platform, with a port for the OMNeT++ simulator, and with an analytic model of network behavior. With the experiments and data collected from various evaluation tools, parameters of the protocol are adjusted and optimized, improving TSTP and taking it one step closer to its goal of being a complete, efficient solution for IoT-ready WSNs.

Keywords: Cross-layer design; Performance evaluation; Communication protocols; Internet of Things; Wireless Sensor Networks

Resumo

Redes de Sensores Sem Fios (RSSF) têm sido implementadas de muitas formas diferentes ao longo dos últimos anos. Prédios, casas, fazendas, rios, o clima, chãos de fábrica, e muitos outros tipos de ambiente, podem todos ser monitorados e controlados por uma rede sem fios composta de dispositivos computacionais baratos equipados com diferentes sensores e atuadores. Com estas redes conectando-se à Internet das Coisas (IoT, do inglês *Internet of Things*), torna-se cada vez mais importante que as mesmas operem de maneira confiável, com sistemas operacionais e protocolos de comunicação orientados ao domínio de aplicação e cuidadosamente implementados.

O *Trustful Space-Time Protocol* (TSTP) é um protocolo *cross-layer* para RSSF projetado para enriquecer dados com semântica de unidades do Sistema Internacional (SI), autenticação, criptografia, temporalidade e georeferenciamento de forma eficiente. Através da integração de dados compartilhados por múltiplos serviços de rede em uma única infraestrutura de comunicação, TSTP é capaz de eliminar a replicação de informação entre serviços, atingindo um sobrecusto modesto em termos de mensagens de controle. Porém, a complexidade das funcionalidades do TSTP, seu amplo escopo - da aplicação ao controle de acesso ao meio, - e sua natureza experimental trazem requerimentos diversos, além daqueles normalmente considerados na maioria dos projetos de *software*.

Nesta dissertação, o projeto de protocolo do TSTP é exposto em detalhes, com uma descrição dos formatos de mensagem e algoritmos usados para controle de acesso ao meio, roteamento geográfico, localização espacial, sincronização temporal e segurança. Então, uma implementação desenvolvida para o *Embedded Parallel Operating System* (EPOS) e a plataforma de RSSF EPOSMote III é apresentada. Para evitar uma implementação monolítica da abordagem *cross-layer*, um projeto baseado em componentes é utilizado, explorando técnicas de metaprogramação com *templates* para adaptar e combinar blocos básicos. Uma arquitetura orientada a eventos que gerencia *buffers* enriquecidos com metadados sem gerar cópias de memória é aplicada para tratar de requisitos transversais. O projeto e a implementação do protocolo são avaliados com experimentos na plataforma EPOSMote III, com um porte para o simulador OMNeT++ e com um modelo analítico de comportamento da rede. Com base nos experimentos e dados coletados por meio de várias ferramentas de avaliação, parâmetros do protocolo são ajustados e otimizados, melhorando o TSTP e trazendo-o um passo mais perto de seu objetivo de prover uma solução completa e eficiente para RSSF integradas à IoT.

Palavras-chave: Design cross-layer; Avaliação de desempenho; Protocolos de comunicação; Internet das Coisas; Redes de Sensores Sem Fios

Resumo Expandido

Introdução

Redes de Sensores Sem Fios (RSSF) têm sido implementadas de muitas formas diferentes ao longo dos últimos anos. Prédios, casas, fazendas, rios, o clima, chãos de fábrica, e muitos outros tipos de ambiente, podem todos ser monitorados e controlados por uma rede sem fios composta de dispositivos computacionais baratos equipados com diferentes sensores e atuadores. Com estas redes conectando-se à Internet das Coisas (IoT, do inglês *Internet of Things*), torna-se cada vez mais importante que as mesmas operem de maneira confiável, com sistemas operacionais e protocolos de comunicação orientados ao domínio e cuidadosamente implementados. Este trabalho avalia e melhora o *Trustful Space-Time Protocol* (TSTP) (RESNER; FRÖHLICH, 2015a; RESNER; ARAUJO; FRÖHLICH, 2017), trazendo-o um passo mais perto de seu objetivo de prover uma solução completa e eficiente para RSSF integradas à IoT.

A IoT destaca desafios que estiveram presentes em RSSF desde o início, mas que puderam ser ignorados até certo ponto em redes e sistemas auto-contidos. Maior atenção deve ser dada à *segurança* da comunicação. Mensagens devem ser autenticadas, com integridade verificável, imunes a *replays*, e às vezes confidenciais. Dados devem ser universalmente interpretáveis em termos de *o quê* está sendo medido, bem como *quando* e *onde* cada medição foi feita. Deve-se ter evidência para *confiar* nos dados. Garantir todos esses requerimentos adicionais pode ter um preço proibitivo para os dispositivos de RSSF de baixo custo se não houver cuidado no projeto e implementação otimizada do Sistema Operacional e do protocolo de rede.

Há uma vasta disponibilidade de trabalhos na literatura para cumprir cada um destes requerimentos. Muitas camadas físicas foram propostas, juntamente com uma miríade de protocolos de roteamento e acesso ao meio (HUANG et al., 2013; PATIL; BIRADAR, 2012). Tais protocolos foram feitos conscientes de energia (LONARE; WAHANE, 2013); estratégias de agregação e fusão de dados foram utilizadas (LEVIS et al., 2004); infraestruturas básicas foram enriquecidas com protocolos de localização (NIAN; SIVA; POELLABAUER, 2017), temporização (DJENOURI; BAGAA, 2016) e segurança (GRANJAL; MONTEIRO; SILVA, 2015); sistemas operacionais foram enriquecidos para dar suporte a abstrações de alto nível (DIXON et al., 2012), juntamente com sistemas de gerenciamento de dados de larga escala (HULBERT et al., 2016). Muitos pesquisadores têm trabalhado com otimizações *cross-layer* para protocolos de RSSF (MENDES; RODRIGUES, 2011).

Aplicações de RSSF muitas vezes requerem muitas ou todas estas funcionalidades. Se estas não são garantidas arquiteturalmente, muitas vezes camadas heterogêneas e auto-contidas de *middleware* são adicionadas ao sistema. Tal adição traz um custo de integração e frequentemente resulta em replicação desnecessária de dados. Um projeto de protocolo *cross-layer* pode eliminar esse sobrecusto, economizar um grande número de mensagens de controle e melhorar os processos de tomada de decisão no protocolo anexando informação de controle em pacotes de dados comuns, subsequentemente organizando e compartilhando tal informação com todos os componentes do protocolo.

O *Trustful Space-Time Protocol* tem o objetivo de prover uma solução completa de comunicação para RSSF/IoT, combinando técnicas para contemplar todos os requerimentos mencionados. O TSTP é um protocolo *cross-layer* orientado a aplicações de RSSF integradas à IoT. O protocolo eficientemente provê funcionalidades recorrentemente necessárias em tais sistemas: dados confiáveis, temporizados, geo-referenciados, conformante com unidades SI, que são eficientemente entregues a um *sink*. O TSTP provê tais funcionalidades diretamente à aplicação na forma de uma entidade completa de rede, o que permite o projeto de cooperações próximas, otimizadas e sinergísticas entre sub-componentes enquanto elimina camadas de software heterogêneas adicionais. TSTP integra sincronização temporal, localização espacial, segurança, controle de acesso ao meio (MAC, do inglês *Medium Access Control*), roteamento e uma API focada nos dados. O projeto de protocolo fortemente acoplado é mapeado em uma implementação modular de baixo sobrecusto, explorando técnicas de metaprogramação de *templates* para adaptar e combinar blocos básicos. Uma arquitetura orientada a eventos que usa *buffers* zero-cópia e metadados é usada para lidar com requisitos transversais.

Objetivos

O principal objetivo deste trabalho é avaliar o desempenho do *Trustful Space-Time Protocol*. Isto é feito por meio dos seguintes objetivos específicos:

- Implementar o TSTP para hardware de RSSF e um simulador de RSSF. As escolhas de implementação serão documentadas, com descrição de técnicas, algoritmos e formatos de mensagens. A implementação deve ser adequada para controlar ambientes físicos do mundo real com sensores e atuadores, bem como disponibilizar os dados gerados para aplicações de mais alto nível;

- Executar experimentos para analizar vários aspectos das implementações. O comportamento do protocolo em termos de precisão da sincronização, latência e consumo de energia será caracterizado para diferentes cenários de aplicação. Os sobrecustos de implementação serão medidos para justificar as técnicas aplicadas;
- Otimizar parâmetros e propor melhorias no protocolo para cada cenário baseado nos resultados das análises.

Metodologia

Neste trabalho, a implementação do TSTP para um dispositivo de RSSF real é apresentada em detalhes. Esta implementação é validada com experimentos controlados, com diferentes ferramentas de análise e *debug* e a observação de diferentes redes TSTP reais. As análises do mundo real são complementadas por experimentos de simulação que investigam cenários diferentes e de maior escala, bem como uma avaliação mais controlada de aspectos específicos. As análises permitem um ajuste fino e a melhoria de diferentes aspectos do protocolo para cada cenário.

Primeiramente, o TSTP foi implementado em C++ no sistema operacional EPOS para a plataforma EPOSMote III, um dispositivo para IoT baseado no *System-on-Chip* CC2538 da Texas Instruments, com um processador ARM Cortex-M3 a 32MHz. A implementação modular do TSTP e uma compatibilidade de linguagem de programação permitiram que o mesmo fosse portado sem muitas alterações do EPOS para o simulador OMNeT++ com o *framework* Castalia. As principais mudanças estão no mediador de interface de rede do EPOS para lidar com o *framework* de rádio do Castalia e no método de notificação do componente `API` do TSTP para interagir com a camada de aplicação do Castalia. Como resultado, o simulador implementa um modelo muito detalhado de uma rede TSTP real, com dispositivos simulados rodando código muito similar ao que é de fato rodado em dispositivos EPOSMote III reais.

Resultados e Discussão

Os principais resultados provenientes das análises de desempenho do protocolo são:

- Os tamanhos de código para ambas as implementações do TSTP (simulador e EPOSMote III), incluindo todas as funcionalidades, são comparáveis a duas implementações de terceiros do protocolo AODV para o mesmo *framework* de simulação. AODV implementa somente roteamento de mensagens;

- O sobrecusto de tempo para alocação de *buffers* zero-cópia no EPOSMote III é 2.31μs. O mecanismo de notificação usado para propagação dos *buffers* é responsável por 11.60% do tempo total de processamento de *buffers* por todos os componentes;

- O código responsável pela sincronização temporal no EPOSMote III é altamente determinístico temporalmente. O atraso entre enviar e receber *Start-of-Frame Delimiters* (SFD) entre dois dispositivos tem uma variação de 93.5ns, incluindo atrasos de *software* e do rádio. Isso leva a um protocolo de sincronização temporal altamente preciso que é capaz de atingir sincronização instantânea de sub-microsegundos, como medido em dispositivos EPOSMote III reais;

- Há um ponto ótimo no tamanho do preâmbulo do MAC que depende da topologia e do tráfego da rede. Selecionando os parâmetros do MAC corretamente, redes de larga escala atingem ciclos ativos dos rádios menores do que 1% enquanto mantém 100% de taxa de entrega de mensagens, com latências médias em torno de 0.5s;

- Incluir leituras de tempo em cada mensagem para habilitar sincronização especulativa de relógios é benéfico em todos os cenários considerados quando comparado a estratégias explícitas de sincronização. Para redes pequenas de tráfego intenso, a redução no número de mensagens leva a um aumento na eficiência energética. Para redes maiores com menor tráfego, a inclusão de leituras de tempo em mensagens de dados melhora levemente as latências e eficiência energética, enquanto mantem um melhor limite inferior na precisão dos relógios da rede;

- Quando a rede não está saturada, o modelo analítico simplista apresenta resultados similares às simulações detalhadas, sendo muito mais rápido para executar. Tal modelo é então um primeiro passo apropriado para analizar o comportamento esperado de novas redes.

O componente de MAC atualmente utilizado pelo TSTP, baseado no RB-MAC, apresenta muitos benefícios: baixos ciclos ativos; resiliência a canais ruidosos, melhorada pela possibilidade de utilização de múltiplos canais sem sobrecusto; habilitar roteamento geográfico reativo e sincronização especulativa; e permitir que dispositivos com maiores restrições energéticas economizem mais energia não retransmitindo mensagens de terceiros. Além disso, o RB-MAC não impõe requerimentos de sincronização de relógio ou escalonamento e estruturamento da rede. Apesar disso e dos resultados positivos das análises em muitos casos, em outros o MAC mostrou mau desempenho em termos de taxa de entrega, eficiência energética e latência, tornando-o inapropriado dependendo dos requerimentos da rede. Como trabalhos futuros para melhoria do MAC, parece promissora a investigação de protocolos síncronos, habilitados pela sincronização de tempo precisa do TSTP, bem como o aproveitamento de conhecimento espacial da estrutura da rede, dado pelas coordenadas TSTP. Outras escolhas de camadas físicas também teriam um impacto significativo nos gargalos de desempenho, visto que a maior parte dos parâmetros do TSTP MAC são especificamente derivados de características de uma camada física IEEE 802.15.4.

Considerações Finais

Este trabalho proporcionou muitas contribuições para o projeto *Trustful Space-Time Protocol*. Apesar de todos os sub-protocolos que fazem parte do TSTP terem existido de alguma forma no passado, esta é a primeira vez que o protocolo como um todo foi detalhadamente documentado, implementado e avaliado. Muitas ferramentas reusáveis diferentes foram desenvolvidas em conjunto com a implementação do protocolo para ajudar no desenvolvimento, depuração, validação e projeto de novas redes. Hoje, o TSTP é utilizado em duas salas automatizadas e em uma rede de estações de monitoramento hidrológico, alimentando uma sofisticada arquitetura de IoT com dados verificavelmente autênticos enriquecidos com semântica de unidades SI, bem como temporização precisa e coordenadas de criação.

Palavras-chave: Design cross-layer; Avaliação de desempenho; Protocolos de comunicação; Internet das Coisas; Redes de Sensores Sem Fios

List of Figures

List of Tables

Contents

1 INTRODUCTION

Wireless Sensor Networks (WSN) have been implemented in many different forms over the years. Buildings, homes, farms, rivers, the weather, assembly lines, and many more physical environments, can all be monitored and sometimes controlled by a wireless network of cheap computing devices equipped with different sensors and actuators. As these networks get connected to the Internet of Things (IoT), it is ever more important that they operate *trustfully*, with carefully-designed and -implemented domain-oriented Operating Systems and network protocols. This work evaluates and improves the Trustful Space-Time Protocol (TSTP) (RESNER; FRÖHLICH, 2015a; RESNER; ARAUJO; FRÖHLICH, 2017), taking it one step closer to its goal of being a complete, efficient solution for IoT-ready WSNs.

1.1 BACKGROUND

The concept of a network of cheap and small devices equipped with a Microcontroller Unit (MCU), sensors, actuators, and wireless communication hardware, is fitting to many application scenarios such as building automation, environment monitoring, industrial coordination, and several others. Usually these networks have at least one master node, called *sink*, towards which data gathered by sensor nodes is routed and ultimately delivered. When the network supports actuation, often the master node is also in charge of triggering command messages to actuators across the network.

Each application scenario has its own priorities in relation to network requirements. An unwatched network of battery-powered sensors monitoring temperature of a remote forest will generate sparse traffic, but need to stay operative for a long period of time with a limited power supply. An industrial assembly-line machine with different coordinated parts might need reliable and low-latency communication with precise time synchronization, while having access to mains power supply. An automated office room might need a more balanced energy/time relation, with some devices such as power-consumption-monitoring outlets with access to mains power generating non-critical messages, while other devices such as alarms or movement sensors are powered by batteries and sparsely generate important messages with a strong time requirement. These issues bring optimization problems involving many variables for the underlying communication protocol to solve for each particular network.

Such optimization problems are not the only challenges that modern WSNs need to consider. In the majority of applications, the most important concern about the network is the *data* collected. The produced data often need to be made available to agents outside the network (such as domain-expert analysts or artificial intelligence) for analysis, who will guide decisions about the physical system of interest, or prompt command messages that, from the perspective of the WSN, come from outside the network. A popular way of making the data available to sophisticated and well-established pieces of software is to connect the WSN to the Internet in some way. This can be done either by using a protocol stack such as TCP/IP to communicate with sensor nodes directly, or by turning the sink into a *gateway* between the WSN and the external world – which often means the Internet. Now, devices once on isolated networks are part of the Internet of Things (IoT).

The IoT highlights challenges that were present in WSNs from the beginning, but that can be ignored to a degree in self-confined networks and systems. Greater attention is brought to communication *security*. Messages need to be authenticated, with verifiable integrity, immune to replays, and sometimes confidential. Data need to be universally interpretable in terms of *what* is being measured, and *when* and *where* each measurement was made. One must have evidence to believe that the data can be *trusted*. Enforcing the fulfillment of all of these additional requirements might take a prohibitive toll on the low-power WSN devices without careful and optimized Operating System and network protocol design and implementation.

There is a vast body of work in the literature to fulfill each of these requirements. Several physical layers have been proposed, along with a myriad of medium access and routing protocols (HUANG et al., 2013; PATIL; BIRADAR, 2012). Such protocols have been made energy-aware (LONARE; WAHANE, 2013); aggregation and fusion strategies have been employed (LEVIS et al., 2004); basic infrastructures have been enriched with location (NIAN; SIVA; POELLABAUER, 2017), timing (DJENOURI; BAGAA, 2016), and security protocols (GRANJAL; MONTEIRO; SILVA, 2015); operating systems have been designed to support higher-level abstractions (DIXON et al., 2012), along with large-scale management systems built to handle the produced data properly (HULBERT et al., 2016). Extensive research is also being carried out on cross-layer optimizations for WSN protocols (MENDES; RODRIGUES, 2011).

WSN applications often require many or all of these features. If the needed functionality is not architecturally granted, often times layers of heterogeneous, self-contained middleware are added. The added software comes with an integration cost and often results in unnecessary replication of data. A cross-layer protocol design can eliminate this overhead, save a large number of control messages, and improve decision making within the protocol by pig-tailing control information on ordinary data packets and subsequently organizing and sharing that information with all protocol components.

Cross-layer designs are highly promising for WSN and wireless communication protocols in general (FU et al., 2014). Duty-cycling protocols at the Medium Access Control (MAC) level are constantly evolving (HUANG et al., 2013), and even the earlier designs demonstrated that devices could operate for a long time off of a pair of AA batteries (POLASTRE; HILL; CULLER, 2004). Network lifetime can be increased even more with energy-aware routing algorithms (OKAZAKI; FRÖHLICH, 2012). Hardware implementations of symmetric-key security algorithms in WSN devices are getting commonplace due to their wide adoption (e.g. AES being included in the IEEE 802.15.4 standard (IEEE..., 2011)).

The Trustful Space-Time Protocol aims to be a complete communication solution for WSN/IoT, combining all of these techniques and more to fully contemplate the aforementioned requirements. TSTP is an application-oriented, cross-layer protocol for IoT-ready WSNs that focuses on efficiently providing functionality recurrently needed by such systems: trusted, timed, geo-referenced, SI-compliant data that is resource-efficiently delivered to a sink. TSTP grants these functionalities directly to the application in the form of a complete networking entity, which allows the design of optimized, synergistic, tight cooperation of sub-protocols while eliminating the need for additional heterogeneous software layers. TSTP integrates time synchronization, spatial localization, security, MAC, routing, and a data-centric API. The tightly-coupled design is mapped onto a low-overhead, modular implementation, exploring template metaprogramming techniques to adapt and combine basic building blocks. An event-driven architecture that makes use of zero-copy buffers and metadata is used to handle crosscutting concerns.

In this work, TSTP's design is detailed with definitions of algorithms, sub-components, and message formats and usage. The implementation of TSTP for a real WSN device is presented in detail. This implementation is validated through controlled experiments, with analysis and debug tools, and observation of different real-world deployments. The real-world analyses are complemented by simulation experiments that investigate different and larger-scale scenarios, as well as more controlled evaluations of specific aspects. The analyses allow fine-tuning and improvement of different aspects of the protocol for each scenario.

1.2 PREVIOUS RELATED WORK BY THE GROUP

This masters project was realized at the Software/Hardware Integration Laboratory (LISHA) at the Federal University of Santa Catarina (UFSC). Over the last decades, several research projects have been conducted in the group under the supervision of professor Antônio Augusto Fröhlich in the topic of IoT and WSN protocols. The present work is built upon the research of many of these previous works. These works granted the group the required perspective and experience which led to the idea of the Trustful Space-Time Protocol. What follows is a list in chronological order of the main past contributions – with their main authors – that directly influenced the present work and ultimately made it possible.

- Gilles Pokam (1999): Pioneering work in static metaprogramming frameworks applied to communication systems;
- Eduardo A. Billo (2002): Application of these techniques on the implementation of a Bluetooth stack;
- Thiago R. C. Santos (2005): Implementation of a component-based communication framework, with buffers circulating from one component to the next;
- Lucas F. Wanner (2006): Application of these techniques on an implementation for WSN devices running EPOS and equipped with the CC1000 radio communication chip;
- Ricardo Reghelin (2006): Design and implementation of HECOPS, a decentralized location system for sensor networks using cooperative calibration and heuristics;
- Rafael P. Pires (2008): Evaluation of HECOPS;
- Lucas Torri (2008): Application of the IEEE 1451 standard on WSN devices running EPOS;

- Tiago R. Mück (2009): Design and implementation of a communication framework for Software-Defined Radio;
- Rodrigo V. Steiner (2010): Performance evaluation of C-MAC, a Configurable MAC framework, on EPOS;
- Alexandre M. Okazaki (2011): Application of energy-aware Ant-Colony Optimizations on routing protocols for WSNs;
- Peterson Oliveira (2012): Application of the Precision-Time Protocol (PTP) on WSN devices running EPOS.

The author would also like to acknowledge the following people and (some of) their direct contributions to the present work:

- Antônio A. Fröhlich: For the idea and high-level design of TSTP, as well as the direct work in every step of TSTP's conception, design, implementation, and documentation;
- Gustavo M. Araujo: For the work and guidance on the initial port of TSTP for the OMNeT++ simulator and the Castalia framework;
- Jean E. Martina: For the expert guidance in the security aspects of the protocol;
- Sérgio A. Soares: For the fruitful early discussions and work on the concrete protocol-level design of TSTP.

1.3 OBJECTIVES

The main goal of this work is to evaluate the performance of the Trustful Space-Time Protocol. This is done by achieving the following specific objectives:

- Implement TSTP for WSN hardware and a WSN simulator. The implementation choices will be documented, with descriptions of techniques, algorithms, and message formats. It should be adequate to control real-world physical environments with sensors and actuators, and make the generated data available to higher-level applications;
- Perform experiments to analyze various aspects of the implementations. The protocol's behavior in terms of synchronization accuracy, latency, and energy consumption will be characterized for

different application scenarios. Implementation overhead will be measured to justify the employed techniques;

- Optimize parameters and propose changes to improve the protocol for each particular scenario based on the results of the analyses.

1.4 METHODOLOGY

TSTP is implemented in two platforms. For most of the performance evaluations, the OMNeT++ (OPENSIM, 2017) network simulator with the Castalia (BOULIS, 2017) framework is used. For real-world applications and evaluation, TSTP is also implemented on the Embedded Parallel Operating System (EPOS), for the EPOSMote III platform. By the time of this writing, several TSTP networks are using this latter implementation for different applications: a smart room at LISHA; a smart room at UFSC's Smart Solar Building (SSB); and a network of hydrologic monitoring stations deployed near UFSC. The use of the protocol in real-world networks helps find problems in the implementation, as well as validate the adequacy of the protocol's features. All the data produced by TSTP nodes is encapsulated with the SmartData API and stored at UFSC's IoT servers (LISHA, 2017).

1.5 OVERVIEW

Chapter 2 presents in detail the cross-layer design of the Trustful Space-Time Protocol. TSTP defines a novel way of thinking and programming WSN: instead of focusing on specific devices with idiosyncratic sensor/actuator hardware, TSTP proposes to think about space-time regions with geographic characteristics represented by units of the International System (SI). Devices in those regions might provide the ability to sense these characteristics and/or modify them. Space-time coordinates are used as network addresses for each device, and therefore algorithms for synchronizing space and time are included in the protocol. Messages are routed geographically as a result of an interaction between a Router and a low-power Medium Access Control (MAC) component. Security aspects of the protocol are also presented in this chapter.

Chapter 3 shows that the tight interactions between TSTP's cross-layer subcomponents do not have to result in a tightly-coupled, monolithic, or overhead-heavy implementation. This chapter brings a detailed presentation of the component-based implementation that explores template metaprogramming techniques to adapt and combine basic building blocks. An event-driven, subscription-based architecture that makes use of zero-copy, metadata-enriched buffers is used to handle crosscutting concerns.

In Chapter 4, several aspects of the protocol are evaluated using different approaches. Experiments with real hardware are performed to evaluate the limits of time synchronization and the correctness of the MAC state machine implementation. Detailed investigations on the impact of several parameters of the protocol are performed on a simulator. This chapter also analyzes the implementation overhead, both in terms of memory and execution time for significative aspects.

The work is concluded in Chapter 5, with a recapitulation of the main contributions and final remarks.

2 TSTP DESIGN

Parts of this chapter appeared earlier in:

- *Design Rationale of a Cross-layer, Trustful Space-Time Protocol for Wireless Sensor Networks*
  (RESNER; FRÖHLICH, 2015a)
- *Speculative Precision Time Protocol: submicrosecond clock synchronization for the IoT*
  (RESNER; FRÖHLICH; WANNER, 2016)
- *TSTP MAC: A Foundation for the Trustful Space-Time Protocol*
  (RESNER; FRÖHLICH, 2016)
- *Design and Implementation of a Cross-Layer IoT Protocol*
  (RESNER; ARAUJO; FRÖHLICH, 2017)

TSTP is composed of a suite of sub-protocols that are intimately integrated in a cross-layer architecture ranging from the Medium Access Control layer to the Application layer. This chapter presents the main insights behind the concept of TSTP. Later, several aspects of *protocol design* are explored for the architecture as a whole and specifically for each of the sub-protocols.

2.1 PRINCIPLES

In general, to fully describe the relevant characteristics of a measurement by a sensor, one needs to at least determine precisely *where* and *when* the measurement happened, and *what* is the quantity being measured. This information might come indirectly, such as "sensor `ABC3217`, with address `1.2.3.4`, reported the value `0x1234` as its 145th measurement." One would then look at some table to figure out where that device with that address is located and when did its 145th measurement happen, then lookup the datasheet of sensor `ABC3217` to determine what `0x1234` means. Most infrastructures will refine and translate this information automatically along its way from measurement to delivery at the sink to storage to display, and finally present useful information to the end user.

TSTP proposes to express information *from the beginning* in terms that make sense in the real world. Every TSTP device is aware of its spatial location, has access to a common source of time, and is aware of what physical quantities it is working with. A measurement from a TSTP device would be, at any point, directly interpretable as “a sensor at coordinates 27°36’01.5”S 48°31’06.4”W reported the value 42kg at December 1st 2017, 14:32:01 GMT”. Unix time presents a well-defined and simple way of representing time. Coordinate systems such as Earth-Centered, Earth-Fixed (ECEF) allow a common representation of space as a triple of coordinates in relation to the center of the Earth. The International System (SI) of units defines a codifiable way of representing physical quantities.

Another major design principle of TSTP is to avoid unnecessary replication of information across the infrastructure. Instead of defining logical addresses to locate devices in the network, TSTP uses Space-Time itself as a network address. This concept might not be applicable to networks in general, but in most cases it is natural to WSNs. Ultimately, one does not want to know “what value is sensor `1.2.3.4` measuring”, but rather “what is the temperature over there now”. A direct addressing by space and time allows the network to naturally route messages geographically, and to eventually detect and drop expired ones. In this context, traditional network addresses are simply an indirect way of representing space.

To further reduce unnecessary replication of information, TSTP leverages the broadcast nature of wireless media to allow nodes to peek at passing traffic and *speculatively* make decisions: when necessary, nodes will look at timestamps and coordinates in TSTP messages to passively synchronize their clocks and location. They might cancel their transmissions by detecting that other nodes transmitted equivalent messages.

As TSTP shifts the focus away from particular sensor nodes and towards data, it becomes even more important to architecturally provide the basis for *trusting* the data. TSTP nodes are able to sign their messages, so that they have verifiable authenticity, integrity, and timing. Nodes are made aware of their sensor hardware accuracy and timing limits, so that they will only respond to data requests when these parameters match. On the advent of aging, broken, or malicious sensors, TSTP provides architectural support for neighbors of an untrustworthy node to speculatively detect a mismatch between what that node says a physical quantity is and what their own sensors are measuring.

To provide a common notion of Space-Time and fulfill the requirements of an IoT-ready WSN, TSTP includes algorithms for spatial localization, time synchronization, Medium Access Control (MAC), power management, routing, and security.

2.1.1 Application Scenario

TSTP primarily supports ad-hoc, multihop networks of wireless devices in which most of the traffic is delivered to a single destination (all-to-sink). Network addresses are replaced by spatial coordinates, such that it benefits applications concerned more with gathering data from a physical environment than with addressing particular network devices. Examples of such applications are:

1. A network of battery-powered devices periodically gathering temperature information from a large open field;

2. A room automation system with network-enabled devices attached to lamps, outlets, and other sensors and actuators, measuring power consumption and controlling it by commanding lights and air conditioning units;

3. An alarm system with different sensors to detect physical intrusion in a region, firing alarms when it does.

TSTP is *not* aimed towards a range of more traditional Internet applications. For example:

1. Applications that handle non-spatial-sensitive data, such as generic file storage services;

2. Applications that handle one-to-many or many-to-many traffic of data, such as video streaming or group messaging;

3. Structured networks mostly concerned with granting high throughput, such as Internet backbone infrastructure.

2.2 ARCHITECTURE

TSTP is designed as a collaboration of 6 protocol components that interact with each other: MAC, Router, Timekeeper, Locator, Security, and API. The **Timekeeper** is responsible for precise clock synchronization across the network. It currently implements the *Speculative Precision Time Protocol* (SPTP) (Section 2.4), but any high-precision protocol could be used. The **Locator** is responsible for keeping spatial coordinates up to date, particularly in nodes devoid of a GPS receiver or for which a static location was not defined at deployment-time. It currently implements the *Heuristic Cooperative Calibration Positioning System* (HECOPS) (Section 2.3). The **Router** implements a greedy, fully-reactive, geographic routing strategy as it decides when to forward messages from other nodes and implements metrics for relay selection used by the MAC (Section 2.5). **Security** is responsible for encrypting and/or signing messages, as well as managing cryptographic keys (Section 2.6). The **MAC** component manages the network interface and communication channel, sending and receiving network packets containing TSTP messages (Section 2.5). The **API** handles interactions between the applications and the network through the protocol (Section 2.7).

Unlike traditional layered architectures, components are designed to interact with one-another to achieve high performance. For example, MAC and Timekeeper collaborate to leverage the low-level synchronization process necessary for demodulation of radio signals, achieving high-precision timestamping and clock synchronization. MAC and Router collaborate to define and apply metrics to select winners in channel access contention. Locator and Timekeeper help Security assess the validity of messages and implement sophisticated time-aware security mechanisms.

To avoid tightly-coupled interactions between component implementations, TSTP employs a vertical data exchange plane model (FU et al., 2014), orderly delivering metadata-enriched zero-copy network buffers to each component, as illustrated in Figure 1. Components share information with each other either by means of metadata attached to each buffer (which is not transmitted through the network), or the TSTP message itself (which may be transmitted through the network) contained in the same buffer. Buffers are further explained in Section 3.1.1.

To prevent replication of information in TSTP messages, instead of confining each component to its own specific header of a message, the full message contained in a buffer is accessible to any component. This way, if more than one component need timing control for messages (e.g. Timekeeper and API), for example, they can all use the same time fields, rather than potentially including the same information (directly or indirectly) more than once. To enable routing and

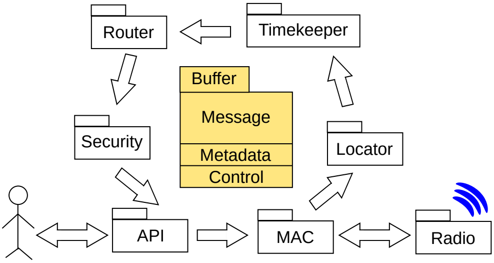


Figure 1: TSTP component interactions and Buffer life-cycle.

passive synchronization, the TSTP common message header includes information about the node that is currently transmitting that message, such as its coordinates and the precise transmission time. The TSTP header is fully presented in Section 3.2.

2.3 POSITION ESTIMATION

The Global Positioning System (GPS) has become the *de facto* standard for automatic device localization. Although it is a reliable solution for outdoor environments, nodes operating indoors generally can't benefit from it, and adding GPS receivers to a WSN node incurs additional energy and financial costs. In mobile wireless networks, there are several algorithms that allow nodes to estimate their location with no additional hardware, leveraging characteristics of particular radio signaling technology, such as the Received Signal Strength Indication (RSSI) provided by an IEEE 802.15.4 implementation (PIRES; WANNER; FRÖHLICH, 2008), or the Time Difference of Arrival (TDOA) provided by a UWB transceiver (OLIVEIRA et al., 2012).

Given that nodes can estimate their distances to one another based on physical layer characteristics and there are anchor nodes in the network that know their own position, there are many techniques to determine the position of a given node, such as trilateration and min-max (NIAN; SIVA; POELLABAUER, 2017). The location of anchor nodes can be determined equipping a subset of the nodes with GPS receivers, or simply pre-set if a node is not mobile. This makes anchor nodes more expensive and/or difficult to deploy, thus it is generally desirable to reduce their number.

Regardless of the chosen position estimation algorithm, it will come with a precision limitation. The Heuristic Environmental Consideration Over Positioning System (HECOPS) (REGHELIN; FRÖHLICH, 2006) enriches the localization strategy by establishing a ranking system to determine the reliability of each estimated position. It uses heuristics to mitigate the effects of measurement errors (which can be high, especially in low-cost nodes). Such enrichments reduce the number of necessary anchor nodes by allowing non-anchor nodes to act as anchors when their confidence is high enough.

The original implementation of HECOPS uses RSSI values on IEEE 802.15.4 devices. Nodes maintain a table of neighbor data containing their alleged coordinates, confidence, and RSSI-based distance measurements. Every node periodically broadcasts its table, and the nodes receiving it can update theirs. With enough data from neighbors, a node n can estimate its position and its confidence C_n via trilateration.

To cope with the irregular nature of RF signals and RSSI (or any other) measurements, HECOPS defines a *deviation* heuristic value, which is obtained when two anchor nodes estimate their distance to each other via RSSI and then compare it to their actual distance (since their coordinates are known). When a deviation is detected between nodes A and B, it is heuristically assumed to affect every node in the triangular area $tri(A, B)$, illustrated in Figure 2.

A node's confidence in its location is determined by Equation 2.1, selecting the 3 reference nodes in its table with the best confidence.

$$C_n = \frac{\sum_{i=1}^{3}(C_i \times 0.75 + C_{tri(i,n)} \times 0.25)}{3} \times 0.8 \tag{2.1}$$

In TSTP, non-anchor nodes build their position estimation tables simply by observing the TSTP header of passing messages, which contain the coordinates and confidence of the transmitting node. The implementation of HECOPS in TSTP, realized by the Locator component, is presented in Section 3.3.

2.4 TIME SYNCHRONIZATION

Time in computing systems is typically kept by counting cycles of a piezoelectric crystal oscillator. The frequency of oscillation is determined by the cut, vibration mode (longitudinal, transverse), and the size of the crystal wafer (ZHOU et al., 2008). Imprecisions and defects in the manufacturing process therefore lead to a deviation in oscillation frequency across different parts with the same nominal frequency.

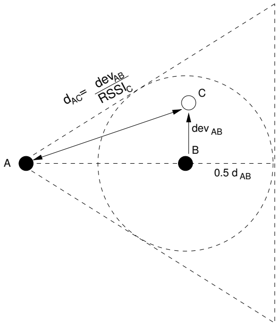


Figure 2: HECOPS deviation heuristic. A and B are anchor nodes that detect a deviation between them. C is inside the $tri(A, B)$ area and compensates for the deviation. Figure from (REGHELIN; FRÖHLICH, 2006)

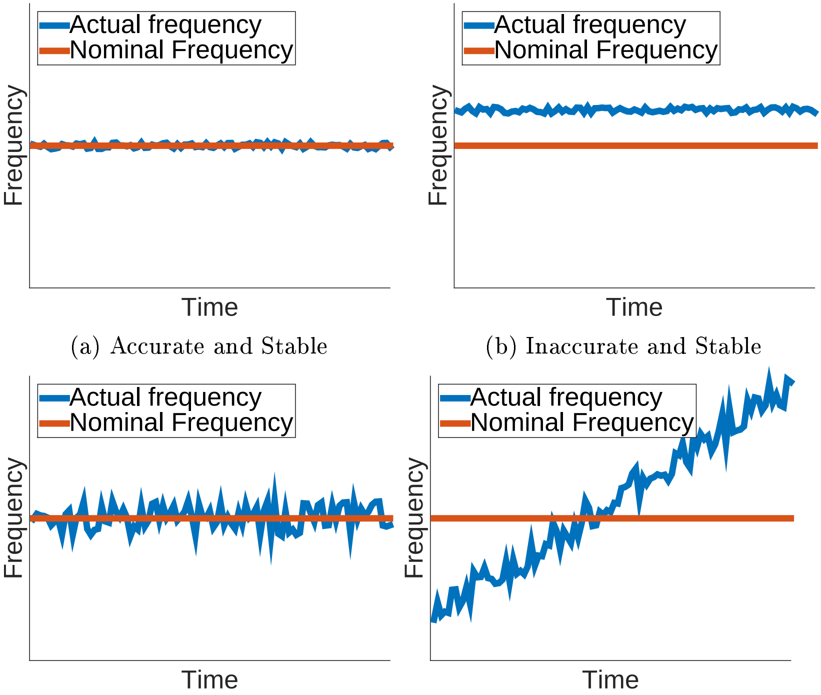
Furthermore, environmental factors such as temperature, aging, drive level, power supply noise, and vibration-induced noise also affect crystal stability and accuracy.

The accuracy of a crystal is its offset from the target nominal frequency, while its stability is the spread of its frequency over time (ZHOU et al., 2008). Figure 3 shows examples for accuracy and stability scenarios for crystal oscillators. Oscillators may be stable and accurate, stable but inaccurate, instable but accurate on the average, and instable and inaccurate. Inaccuracy in frequency leads to the fact that two independent clocks, once synchronized, will drift apart without limit. Figure 4 shows the clock readings of different EPOSMote III devices drifting apart over time[1].

Improving stability and precision of clocks has been the target of research and development, including Temperature-Compensated Crystal Oscillators (TCXO), Microcomputer-Compensated Crystal Oscillators (MCXO), and Oven-Controlled Crystal Oscillators (OCXO), all of which attempt to compensate for systematic and environmental variations in oscillators (LEWIS, 1991). These mechanisms come at a cost in terms of system volume, cost, and energy consumption. A complementary option is the employment of network time synchronization protocols.

Network time synchronization protocols typically work through

[1] Readings from CC2538's 32MHz timer with ±40ppm accuracy dedicated to network software.



(a) Accurate and Stable

(b) Inaccurate and Stable

(c) Accurate and Unstable

(d) Inaccurate and Unstable

Figure 3: Accuracy and Stability of Crystal Oscillators

the exchange of timestamped messages between pairs of nodes (CRISTIAN, 1989). Each message typically contains a local timestamp of the sender. A series of messages are exchanged to estimate the time offset and drift between a pair of nodes (GUSELLA; ZATTI, 1989). The most widely used protocol of this kind is the Network Time Protocol (NTP) (MILLS, 1991). NTP targets Internet hosts and, due to its multihop nature, suffers from constantly varying communication delays between hosts. Typical synchronization accuracy for NTP is on the order of tens of milliseconds, making it unsuitable for time-critical sensing and actuation applications.

In the wireless embedded sensing context, the Flooding Time Synchronization Protocol (FTSP) (MARóTI et al., 2004) is the most widely cited time synchronization protocol. FTSP aims to synchronize an entire multihop network of wireless nodes to a single root node. The root node periodically broadcasts time synchronization messages,

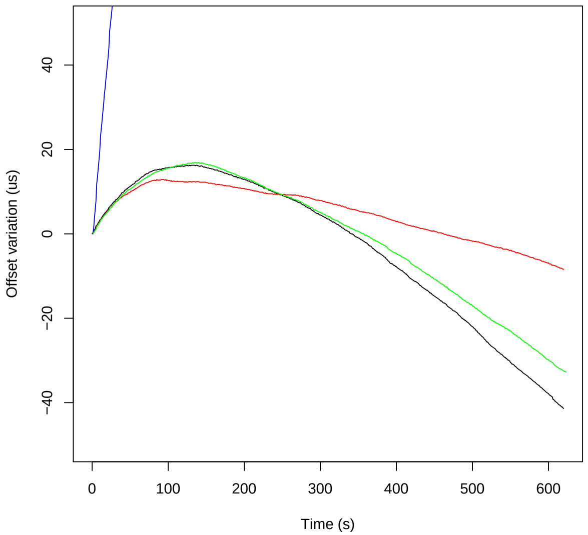


Figure 4: Clock drift of four different EPOSMote III devices in relation to a fifth. One of the motes (in blue) diverged particularly quick in this scenario.

each of which contains multiple timestamps. FTSP targets platforms using radios with software-defined medium access control (MAC), and relies on MAC-level timestamping combined with a characterization of interrupt handling timing. Each transmitted timestamp is acquired as close to the physical send event as possible. This combination of precise interrupt handling timing and multiple timestamps per message allowed FTSP to achieve a 1-hop synchronization accuracy of ~1.5 μs.

The introduction of the IEEE 1588 standard (IEEE, 2008), the Precision Time Protocol (PTP), emphasized the need for high precision, sub-microsecond synchronization for distributed sensing and control systems (EIDSON, 2006). Compared to NTP, which uses application-level timestamping of network packets to account for drift and offset, PTP can make use of advanced timestamping capabilities in the network interface hardware in order to reduce the temporal interference introduced by various layers of software. PTP defines a hierarchy of clocks, in which a grandmaster periodically broadcasts `Sync` messages, and other nodes send `Delay Request` messages which are replied by higher clocks in the hierarchy, so that each node can measure with high accuracy both network round-trip time and their own clock offset.

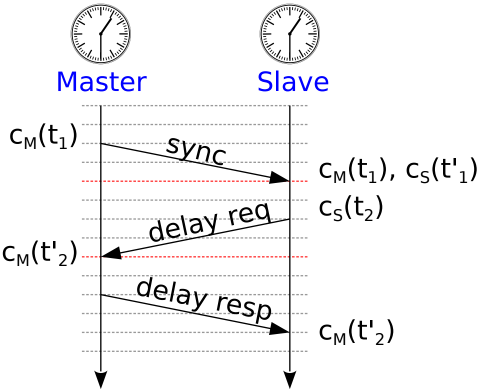


Figure 5: Example of clock synchronization with PTP. $c_M(t_1)$ and $c_M(t'_2)$ are transmitted with the `sync` and `delay resp` messages, respectively.

To successfully synchronize in time, a node needs to estimate its clock *offset* and *frequency drift* in relation to a reference clock. Let M denote a master node with a reference clock and S a slave. Let $c_M(t_m)$ represent the value of the timestamp counter of node M at physical time t_m, and $c_S(t'_m)$ denote the value of the timestamp counter of node S when it receives a message containing $c_M(t_m)$. By collecting 4 timestamps over the exchange of 3 messages, as illustrated in Figure 5, a slave node is able to estimate the round-trip time d_{TX} according to Equation 2.2, and adjust its clock by subtracting the offset ϕ given by Equation 2.3.

$$d_{TX} = \frac{(c_M(t'_2) - c_S(t_2)) + (c_S(t'_1) - c_M(t_1))}{2} \tag{2.2}$$

$$\phi = c_S(t'_1) - (c_M(t_1) + d_{TX}) \tag{2.3}$$

Once the offset is corrected, node S is ready to estimate its clock frequency drift in relation to M. After it receives a new message containing $c_M(t_3)$, the clock drift is given as:

$$\hat{f}_e = \frac{(c_M(t_3) - c_S(t'_3)) - (c_M(t_2) - c_S(t'_2))}{c_M(t_3) - c_M(t_2)} \tag{2.4}$$

The accuracy of this estimation is (SCHMID; DUTTA; SRIVASTAVA, 2010):

$$\delta_Q = \frac{1}{(t_3 - t_2) \cdot f_M} \tag{2.5}$$

where f_M is the frequency of the clock of node M. The offset and drift estimation should be continually re-estimated, as clocks (especially the cheap clocks present in most WSN platforms) usually drift semi-randomly over time.

The set of equations presented allows synchronization even if both nodes are multiple hops away, and do not make assumptions about the underlying network[2]. However, protocols such as FTSP require that the time d_{TX} that timestamps take to travel from Master to Slave is predictable with small enough jitter. If that is the case, calculating clock offsets become trivial, as Equation 2.3 can be applied simply by observing one timestamp in the network. By observing a second timestamp, drift can be estimated according to Equation 2.4.

The Speculative Precision Time Protocol (SPTP), developed and used in the TSTP context, defines that synchronization happens hop-by-hop, and nodes synchronize their clocks with any other node that is closer to the sink, which is the ultimate reference. By restricting synchronization to neighboring nodes, leveraging low-level timestamping by the MAC, and timestamping every message, SPTP estimates d_{TX} to high precision and allows nodes synchronize their clocks completely passively, just by observing network traffic, while keeping synchronization accuracy close to the maximum possible offered by the hardware platform. If a node is in a region with too sparse traffic, it can generate `Keep Alive` messages, which carry only the TSTP header and prompt a response from neighboring nodes to collect synchronization information. It can also set a `Time Request` bit on its messages to trigger responses from neighbors. Figure 6 shows an example where node A sends a data message towards the sink, and synchronizes its clock to node B while its message gets routed. SPTP's implementation is shown in Section 3.4.

2.5 MAC AND ROUTING

By including spatio-temporal data in every message, TSTP devices are able to localize themselves mostly passively, just by overhear-

[2] PTP allows "follow up" messages or low-level timestamping to improve synchronization accuracy, but these techniques do not change the fundamentals expressed in the equations.

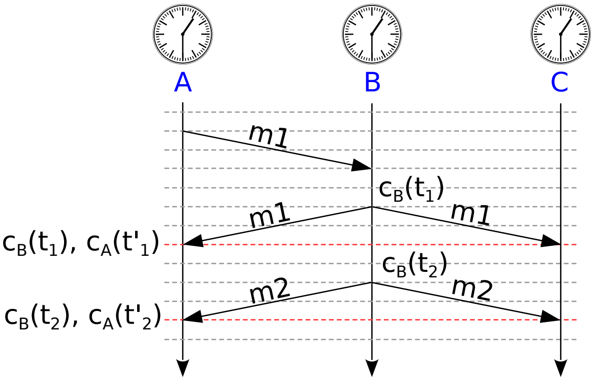


Figure 6: Example of clock synchronization between A and B with SPTP. Nodes to the right are closer to the sink. m1 and m2 are regular data messages being routed towards the sink. m2 can also be triggered by setting the `Time Request` bit in m1.

ing network traffic. The overhearing of traffic is possible because the wireless medium is inherently broadcast, such that any radio transmission can be received by any listening device in a certain area[3]. This broadcast nature brings problems of sender coordination: if more than one transmission happens at the same time, they might collide and destroy one another. Furthermore, radio hardware is relatively power-hungry in WSN devices, and much energy can be wasted by active radios listening to the channel when there is no ongoing transmission, i.e. idle listening (POLASTRE; HILL; CULLER, 2004).

WSN protocols usually define duty cycling techniques to mitigate this problem: radios are turned on periodically to quickly check the channel, and then turned off if no activity is detected. This brings a toll on senders, however, since they do not know if there will be any receiver awake at all when they transmit their message. The classic solutions to this problem are either synchronizing and scheduling transmissions, or sending long preambles (proportional to the sleep period) before each message, to make sure that receivers will wake up at some point during the preamble and detect the transmission. Protocols adopting this latter solution are categorized as Preamble-Sampling MACs. A perfect, global synchronization is generally a better approach, but very

[3] Calibrated with compatible frequency and demodulation parameters, and in a location and moment where the transmitted signal can arrive with enough strength and without too much destructive interference.

hard to achieve in practice, and can incur large overhead in terms of control messages. Preamble-sampling is generally much simpler, but may impose large overhead in the transmission of any message.

B-MAC (POLASTRE; HILL; CULLER, 2004) is a well-known example of the Preamble-Sampling approach. Transmitters send a long continuous preamble and when a receiver detects it, the radio is kept on until reception of the data. In other protocols, such as MFP (BACHIR et al., 2006), the continuous preamble is divided into a series of small frames, called *microframes*, containing a sequence number which serves as a countdown to the actual data transmission. This allows receivers to receive a single microframe, switch off its radio during transmission of the remaining microframes, and switch it on again just before the data frame transmission.

In the Receiver-Based MAC (RB-MAC) (AKHAVAN; WATTEYNE; AGHVAMI, 2011) protocol, senders transmit data without defining a MAC-layer destination. Preambles consist of microframes that contain useful information such as countdown to data transmission, sender distance to sink and payload sequence number. All neighboring nodes within communication range of the sender sense the channel every S interval, obtain a microframe and extract the information; then, only eligible receivers (nodes closer to the final destination of the message) go back to sleep and wake up to receive the data at the time indicated by the countdown. Nodes that receive the data without error are relay candidates, and start a contention timer based on its own distance to the destination and possibly other factors (e.g. remaining energy), which when elapsed will trigger a CCA. The node with the shortest contention time offset will sense no channel activity and proceed to transmit the preamble for S units of time. Other relay candidates will detect the winner's preamble containing the same sequence number, drop the data and go back to sleep since the packet is already being forwarded. Figure 7 illustrates this process. As a consequence of the way relay candidates are determined, packets are geographically routed to the final destination in a greedy way.

RB-MAC is significantly more resilient to lossy links when compared to sender-based MAC protocols (in which sender nodes keep the addresses of perceived neighbors and define a specific neighbor as receiver for each message) (STEINER et al., 2013). As the number of network nodes increases, RB-MAC requires fewer retransmissions, consequently reducing latency and energy consumption (STEINER et al., 2013).

TSTP MAC is the component responsible for interfacing TSTP

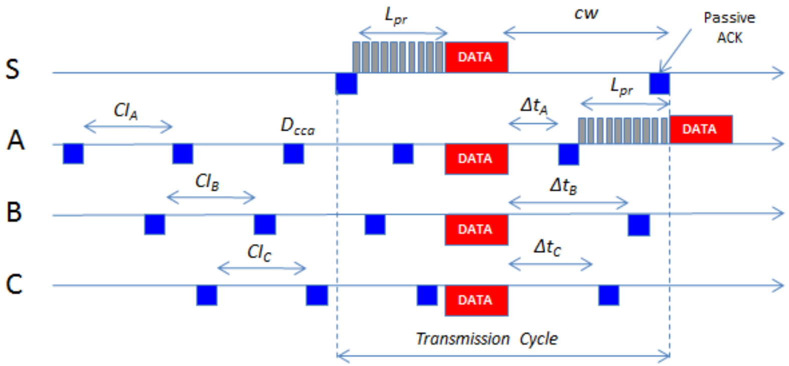


Figure 7: Example of an RB-MAC message forwarding. Figure from (STEINER et al., 2013).

with the network with a special care for energy consumption. It currently implements the mechanisms of RB-MAC. The contention offset δ for a message m at any given node is locally calculated as:

$$\delta(m) = \frac{R - (D_m - D)}{R} \times S \tag{2.6}$$

with D representing the current node's distance to the message's destination, D_m representing the message sender's distance to the destination, and R representing a network-wide parameter corresponding to the radio range of the nodes. This equation makes nodes closer to the destination wake up earlier, normalizes the offset to the period S (since nodes only route messages that come from nodes more distant to the destination – as explained in Section 2.5.2, – it holds that $D < D_m$), and ensures that, if the destination is extremely far away, neighboring nodes (that are at the same discrete point in space in the scale of the total distance) will still have different offsets.

Messages being routed are kept on each involved node i in a queue Q_i. Each entry e_Q in Q_i represents a message m that is scheduled for transmission or retransmission. In addition to the message, e_Q also holds m's `Id` (extracted from the microframes that preceded it), its `Expiry` (extracted from its header), its destination `Dst` (either a previously assigned sink or extracted from the payload), and the contention offset δ.

TSTP uses RB-MAC's implicit acknowledgment (ACK) to confirm the routing of messages. A node *only* removes a message from its queue when it expires or when another message with the same `Id` is overheard in the network (that is, when another node handles the forwarding of that message towards its destination). The only case in which an explicit ACK is used is when the message reaches its final destination: that node must retransmit the same message, just to acknowledge the last forwarder and any neighbors that might still have that message queued. Since this last ACK has a `Last Hop` coordinate identical to the destination, the calculated distance of zero won't be matched by any other node in the vicinity. The rest of this section concerns routing aspects integrated in TSTP MAC. Further implementation details of the MAC are exposed in Section 3.5.

2.5.1 Spatial Distortion

The default distance-based contention offset δ, given by Equation 2.6, can be made sensitive to other routing metrics according to application's needs. For example, it may take into account the remaining battery charge or buffer space of a node to ensure that nodes in an optimal path are not going to have their batteries depleted too quickly (OKAZAKI; FRÖHLICH, 2012). To accomplish this, TSTP uses the notion that those additional metrics *distort space*. A node running out of memory or consuming too much energy can *stretch* space, increasing its distance to the destination so that other nodes in the vicinity become more likely to win the contention to retransmit a message. Conversely, a node transmitting a message that is close to expiring can *shrink* space, reducing its distance to the destination and thus increasing the chances of winning the contention and getting the message transmitted earlier. Equation 2.7 redefines the offset equation introducing a distortion coefficient $\alpha \in [0, 1]$:

$$\delta(m) = \alpha \times \frac{R - (D_m - D)}{R} \times S \tag{2.7}$$

The distortion coefficient α defines how much other metrics influence the perceived distance, and hence the offset used for contention. It may take into account virtually any metric of interest, as long as it is a real number in the interval $[0, 1]$. A value of $\alpha < 1$ decreases δ, increasing the node's likelihood of winning the contention, while $\alpha = 1$ takes into account only the node's distance to the destination.

2.5.2 TSTP Greedy Forwarding Algorithm

To prevent the distortion coefficient from causing messages to be forwarded to an incorrect destination, the TSTP Greedy Forwarding Algorithm (Algorithm 1) ensures that all messages that a node receives and are queued for transmission satisfy the Progress Property: all messages relayed will make positive spatial progress towards the destination. This property can be written as $\forall j \forall i \{m_j i \in Q_i | D_i < D_j\}$, meaning that each message $m_j i$ from node j overheard by node i will be stored in node i's transmission queue Q_i if and only if the distance D_i from node i to the message's destination is smaller than the distance D_j from node j to the same destination.

Algorithm 1 TSTP Greedy Forwarding Algorithm

```
 1: procedure GreedyForward($m$)
 2:     $queued \leftarrow false$
 3:     for each $e_Q \in Q_i$ do
 4:         if $m.id = e_Q.id$ then
 5:             $queued \leftarrow true$
 6:             if $Distance(m.LastHop, m.Dst) > Distance(Here, m.Dst)$ then
 7:                 delete $m$
 8:             else
 9:                 $Q_i$.remove($e_Q$)
10:                 delete $m$
11:                 delete $e_Q$
12:             end if
13:         end if
14:     end for
15:     if $queued = false$ then
16:         if $Distance(Here, m.Dst) > Distance(m.LastHop, m.Dst)$ then
17:             delete $m$
18:         else
19:             $Q_i$.insert($m$)
20:         end if
21:     end if
22: end procedure
```

Algorithm 1 handles four possible cases, illustrated in Figure 8:

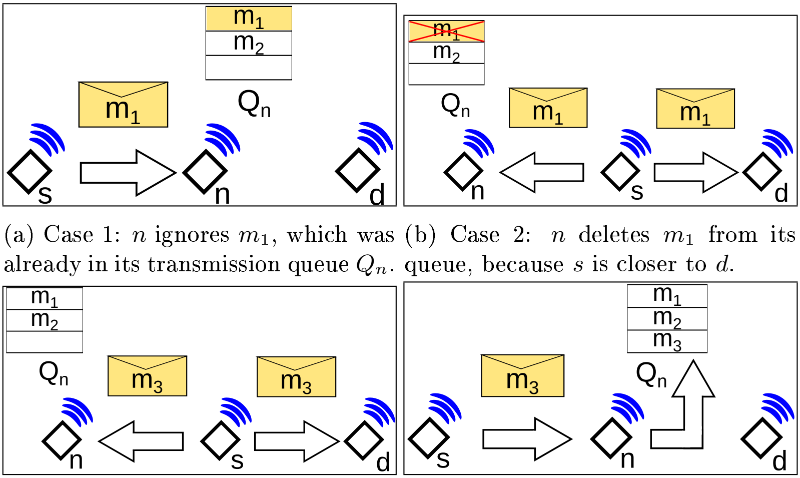
Case 1 (lines 6, 7) : If a copy of the message m is already queued and m is coming from a node farther from the destination, then

m is a retransmission attempt and can be ignored (the local copy remains on the queue for later retransmission).

Case 2 (lines 8, 9, 10, 11) : If a copy of m is already queued and m is coming from a node closer to the destination, then this means that the message has already made positive progress to the destination. The message m is therefore handled as an acknowledgment, causing the local copy e_Q to be removed from Q_i.

Case 3 (lines 16, 17) : If m is a new message but came from a node closer to the destination, then it means that this node (the receiver) would not make positive progress towards the destination. The message is ignored.

Case 4 (lines 18, 19) : If m is a new message that came from a node more distant from the destination, then this node can make positive progress towards the destination. It stores the message on its queue and becomes a relay candidate for message m.



(a) Case 1: n ignores m_1, which was already in its transmission queue Q_n.

(b) Case 2: n deletes m_1 from its queue, because s is closer to d.

(c) Case 3: n ignores m_3, because s is closer to d.

(d) Case 4: n adds m_3 to its queue, because n is closer to d than s is.

Figure 8: Possible cases for Algorithm 1. In each case, node s sends a message with d as the destination, which is overheard by node n.

The following theorem proves that the Greedy Forwarding Algorithm ensures the Progress Property:

Theorem 1. *The Greedy Forwarding Algorithm will store in queue Q_i any given incoming messages m_ji from node j overheard by node i if and only if $D_i < D_j$.*

Proof. The proof is by contradiction. Suppose that $\exists i, \exists j, \exists m_ji\{m_ji \in Q_i | D_i \geq D_j\}$ and that every Q is only altered by Algorithm 1. The message m_ji must have been included in Q_i by Case 4 because it is the only case that includes messages in the queue. But Case 4 only includes a given message m_ji in Q_i if $D_i < D_j$, and since by assumption $D_i \geq D_j$, it is not possible that m_ji was included in Q_i by Case 4. Therefore, since Case 4 is the only case that could include m_ji in Q_i, it must be true that $m_ji \notin Q_i$, reaching a contradiction. □

The theorem assumes that distances are expressed correctly. Distances may be incorrect either by deliberate manipulation by a malicious node or by imprecision in the localization algorithm. For the first case, since the router takes as parameter the average radio range R of the network (Section 2.5), nodes drop messages that allegedly come from a distance greater than R. Besides preventing undesirable effects of radio range asymmetry, this mechanism results in only nodes one hop away from the alleged source coordinates of the malicious message trying to forward it, and from that point on the message will be routed correctly. Regarding imprecisions in the localization algorithm, it may break the theorem at hops where nodes are so close that their relative positions are inverted. However, this will only ultimately route the message to an incorrect destination if the locations of all nodes are consistently and heavily skewed in the wrong direction, which is a very unlikely situation. In both cases the message can take non-optimal hops towards the destination, but will only be routed to an incorrect final destination if a large portion of the network is heavily compromised.

2.6 SECURITY

The fact that any properly configured radio interface can monitor or participate in WSN communications is very convenient for attackers. Generally, a secure WSN protocol must consider the principles of *confidentiality*, *authenticity* and *integrity* (SUO et al., 2012). AES is a block cipher that can enable these principles while being suitable to low-power WSN devices. In fact, AES is a popular option in this domain, being included in the IEEE 802.15.4 standard (IEEE..., 2011), and many devices come with efficient hardware-accelerated AES engines.

AES requires symmetric keys to be shared between the communicating parties, and other strategies and/or algorithms are needed to solve the key establishment problem. Loading a symmetric key in persistent memory at the sensor device before deployment is the trivial solution for key establishment. This method is unsuitable to many application scenarios, however, since leakage of the pre-loaded key grants third parties the ability to read all private communication (past and future), and act as the node with the stolen identity.

The use of Elliptic Curve Cryptography is popular for establishing shared keys over untrusted channels because of its good processing/security trade-off, making it suitable for resource-constrained devices. There are many efficient implementations (LUK et al., 2007) (KARLOF; SASTRY; WAGNER, 2004) and proposals (HUANG et al., 2003) (FRÖHLICH; STEINER; RUFINO, 2011) for security schemes in WSN and IoT, but they usually require either that a third-party acts as a Certificate Agent using a secure, out-of-band channel, or that sensitive cryptographic information (e.g. a pre-set secret) is pre-loaded in the sensor node.

TSTP's security strategy, outlined in Figure 9, contrasts by minimizing the pre-deployment effort, utilizing unique sensor IDs, synchronized clocks, and time and place of deployment as naturally shared information between a sensor node and the sink. It takes advantage of hardware-accelerated AES engines for encryption, key derivation, and One-Time Passwords (OTP) generation using Poly1305-AES, an AES-based Message Authentication Code which computes a 16-byte authenticator of a variable-length message using a 16-byte AES key, a 16-byte additional key, and a 16-byte nonce (BERNSTEIN, 2005).

The protocol assumes synchronized clocks, and timestamps are truncated to a suitable time window, taking into account the clock synchronization accuracy. Sensor nodes are assumed to hold a unique identifier (ID) known only by them and the sink, and $Auth$ is calculated independently by both as a one-way hash function of the ID. Each party also holds an Elliptic Curve Diffie-Hellman (ECDH) public-private key pair.

Figure 10 illustrates TSTP's security two-way handshake process. The first step for mutual authentication and key establishment between a sensor and the sink is a regular ECDH agreement, which will result in a shared Master Secret K_{ms}. This process is started by the sink, which sends an `ECDH Request` message. Upon reception, the

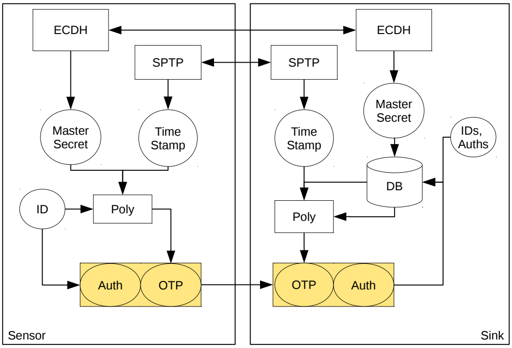


Figure 9: TSTP key bootstrapping overview.

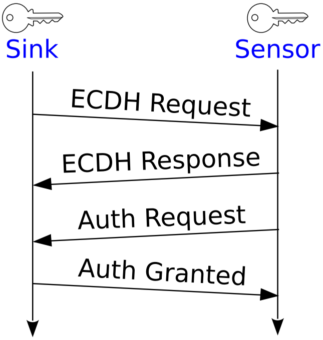


Figure 10: TSTP key bootstrapping message exchange.

sensor node sends back an `ECDH Response` containing its own public key.

Afterwards, the sensor node calculates a One-Time Password using the Poly1305-AES algorithm, according to Equation 2.8, where T is the current truncated timestamp. The calculated OTP is then sent along with the *Auth* code to the sink in an `Auth Request` message. The OTP proves knowledge of both the sensor's ID and the shared Master Secret.

$$OTP = Poly_{1305}(K_{ms}, ID, T) \tag{2.8}$$

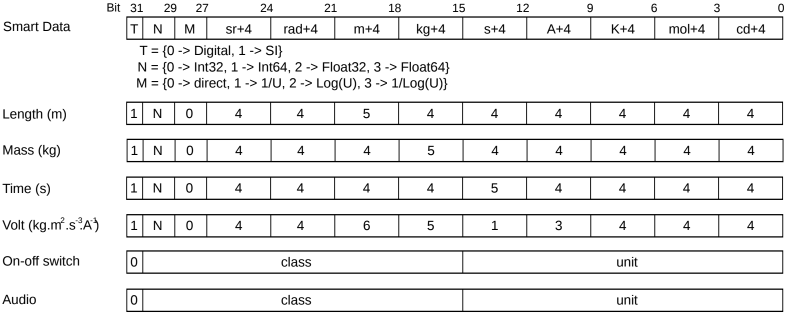
For authentication, the sink fetches on its database the corresponding ID for the received *Auth* and reproduces the OTP calculation for every pending K_{ms}, until a match is found – in which case the matching ID and K_{ms} are tied together, and the sink has evidence that K_{ms} was shared with the only legitimate holder of that particular ID. The sink proceeds by sending back an `Auth Granted` message containing the `Auth OK` code, which is the *Auth* encrypted under a fresh OTP. If the sensor node can decrypt this message and find its *Auth*, it has evidence that K_{ms} was shared with the only other legitimate node that knows its ID: the sink. From this step on, secure messages are signed with a MAC, and padded and encrypted with AES using a fresh OTP as key, assuring data confidentiality, authenticity, integrity and temporality.

The implementation of the security mechanisms described is discussed in Section 3.6.

2.7 SMARTDATA

SmartData was conceived to be the primary (if not the only) abstraction used by application programmers to interact with the physical world on a network of sensors and actuators. A SmartData is a piece of data enriched with enough metadata to make it self-contained regarding semantics, spatial location, timing, and trustfulness. The semantic aspects of a piece of SmartData is described using a strategy inspired by the Transducer Electronic Data Sheets in the IEEE 1451 standard (INSTRUMENTATION; SOCIETY", 2007). Each piece of data is tagged with a 32-bit type identifier designating either an *SI Physical Quantity* or plain digital data. Physical quantities are identified by the corresponding SI Unit as illustrated in Figure 11. Derived SI units are expressed in terms of SI basic units, as for instance Volt in the Figure. Digital data are simply classified at this level, with actual encoding being specific to each defined class.

The SmartData interface is depicted in Figure 12. Although straightforward and easy to deploy, this interface encapsulates a complex and powerful mechanism to abstract any sort of sensor and actuator on the network. Instances of SmartData can be created using one of two constructors: the first one is used to abstract local transducers, while the second is used to create local proxies of remote transducers. In either case, the binding of a SmartData object with the corresponding transducer is done via the `Transducer` class parameter. Every transducer is supposed to declare a constant named `UNIT`, initialized following the scheme presented in Figure 11. SmartData uses that constant to personify the corresponding SI quantity. For instance, a SmartData object instantiated with a transducer specifying K (Kelvin) as `UNIT`



Bit	31	29	27	24	21	18	15	12	9	6	3	0
Smart Data	T	N	M	sr+4	rad+4	m+4	kg+4	s+4	A+4	K+4	mol+4	cd+4
Length (m)	1	N	0	4	4	5	4	4	4	4	4	4
Mass (kg)	1	N	0	4	4	4	5	4	4	4	4	4
Time (s)	1	N	0	4	4	4	4	5	4	4	4	4
Volt (kg.m^2.s^{-3}.A^{-1})	1	N	0	4	4	6	5	1	3	4	4	4
On-off switch	0	class						unit				
Audio	0	class						unit				

T = {0 -> Digital, 1 -> SI}
N = {0 -> Int32, 1 -> Int64, 2 -> Float32, 3 -> Float64}
M = {0 -> direct, 1 -> 1/U, 2 -> Log(U), 3 -> 1/Log(U)}

Figure 11: SmartData SI unit encoding.

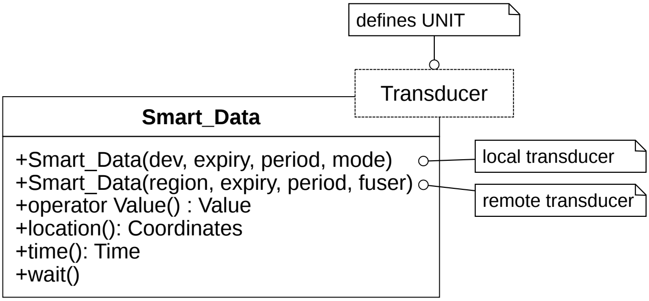


Figure 12: SmartData interface.

represents the SI quantity Temperature. This SmartData can abstract either a temperature sensor or an air conditioner (i.e. a temperature actuator). An accelerometer would be exported as a SmartData representing the SI quantity Acceleration bound through the SI unit $m.s^{-2}$.

SmartData instances also interface with TSTP, associating local instances to Interest messages from the network, and triggering sensor readings at the appropriate times to produce Response messages.

3 TSTP IMPLEMENTATION

Parts of this chapter appeared earlier in:

- *Speculative Precision Time Protocol: submicrosecond clock synchronization for the IoT*
  (RESNER; FRÖHLICH; WANNER, 2016)
- *TSTP MAC: A Foundation for the Trustful Space-Time Protocol*
  (RESNER; FRÖHLICH, 2016)
- *Design and Implementation of a Cross-Layer IoT Protocol*
  (RESNER; ARAUJO; FRÖHLICH, 2017)

A cross-layer protocol design can eliminate a large number of control messages and improve decision making within the protocol by pig-tailing control information on ordinary data packets and subsequently sharing that information among the components of the protocol stack. In this chapter, the software architecture that was used to implement TSTP's cross-layered design on the *Embedded Parallel Operating System* (EPOS) (LAB, 2017) avoiding a monolithic, tightly-coupled software is discussed. Template metaprogramming techniques are applied to implement a component-based, event-driven architecture that efficiently moves messages stored in zero-copy buffers enriched with cross-layer metadata across protocol components at the same time as it eliminates unnecessary dependencies. The implementation of each sub-component is also presented in detail.

3.1 COMPONENT MODEL

TSTP is implemented as a collaboration of 6 protocol components that interact to implement its cross-layer design: MAC, Router, Timekeeper, Locator, Security, and API. The **Timekeeper** is responsible for precise clock synchronization across the network. It currently implements the *Speculative Precision Time Protocol* (SPTP) (RESNER; FRÖHLICH; WANNER, 2016), but any high-precision protocol could be used. The **Locator** is responsible for keeping spatial coordinates up to date, particularly in nodes devoid of a GPS receiver or for which a static location has not been defined at deployment-time. It currently implements the *Heuristic Cooperative Calibration Positioning*

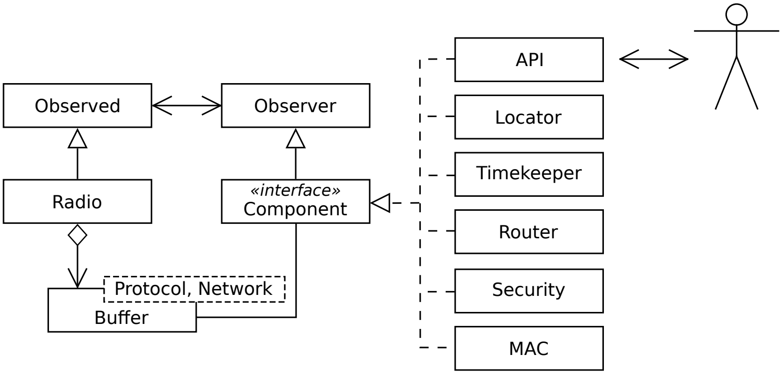


Figure 13: TSTP architecture overview.

System (HECOPS) (REGHELIN; FRÖHLICH, 2006). The **Router** implements the greedy, fully-reactive, geographic routing used by TSTP as it defines and implements the metrics for relay selection used by the MAC when forwarding messages (RESNER; ARAUJO; FRÖHLICH, 2016). **Security** is responsible for the encryption and authentication of messages and also for the management of cryptographic keys (RESNER; FRÖHLICH, 2015b). The **MAC** component manages the communication channel, sending and receiving network packets containing TSTP messages (RESNER; FRÖHLICH, 2016). The **API** handles interactions between the applications and the network through the protocol[1]. Figure 13 illustrates this architecture.

TSTP messages are stored in zero-copy buffers, which aggregate metadata that is used by the components in the protocol while interacting with each other, avoiding explicit function calls between components. Buffers circulate protocol elements in a fixed ordering defined around the six component categories. On receiving, buffers circulate from MAC to API, passing by Locator, Timekeeper, Router, and Security. On sending, they follow the path from the application to API to MAC in the same order, but with a second visit to the MAC. On the first visit to the buffer, MAC populates headers, basic metadata, and takes care of the data provided by the application. The buffer then goes through all the components for processing (for example, signing and encrypting by Security) before returning to MAC for injection into

[1] Interfaces such as SmartData interact with the API component and are seen as application by TSTP.

the network via the Radio hardware mediator[2]. This cycle is depicted in Figure 1, and detailed in Section 3.1.3.

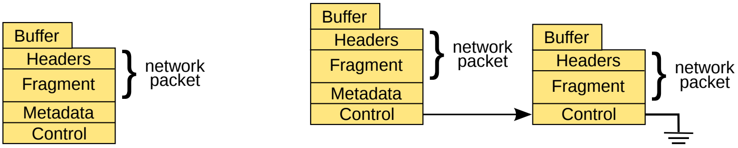
3.1.1 Zero-copy Buffers

For the implementation of TSTP, EPOS *Zero-Copy Buffers* (SANTOS; FRÖHLICH, 2005) were extended. These buffers were first devised aiming at high-throughput, low-latency, low-overhead message exchange between components, but the resource-constrained scenario in which many IoT devices operate also requires the memory to be dynamically allocated with parsimony. As in the original version, data structures are kept in a ring buffer with ownership transfered from one component to another while hiding unnecessary information. However, differently from the original version, the whole ring buffer is not allocated at initialization-time. Buffers are allocated from an optimized system's heap on demand. This heap avoids much of the overhead of application-level heaps by assuming allocations and deallocations at system-level are never malicious and thus storing all the needed control information along with the allocated memory chunk.

To receive messages from the network, TSTP allocates a zero-copy buffer passing information about the maximum payload size, the combined size of the non-recurring protocol headers, and the size of the header that must be repeated in case the underlying network requires fragmentation. Allocation takes place considering the Maximum Transmission Unit (MTU) of the underlying networks, and the result is a complex object with methods to access any of the elements in the message. Indeed, the buffer is modeled as a parameterized class that takes both the protocol and the underlying network as parameters to perform optimized allocations. On IEEE 802.15.4 networks, TSTP does not require nor allows fragmentation, so the resulting zero-copy buffer structure is the one depicted in Figure 14a. Other protocol/network arrangements would use the same interface to handle an object like the one in Figure 14b. The allocated buffer is shared with the network adapter for DMA, and an interrupt is used to signalize that a new packet has been received[3].

[2] In EPOS, hardware mediators realize interfaces for hardware interaction, abstracting most platform-dependent elements. Similar to device drivers.

[3] EPOS zero-copy buffers support network adapters that can handle multiple packets asynchronously. In this case, the whole ring buffer is shared with the adapter, which has full access to the memory bus and utilizes an atomic, per-buffer lock shared with the main processor to synchronize concurrent accesses. This



(a) Buffer without fragmentation. (b) Buffer with fragmentation.

Figure 14: EPOS zero-copy buffer optimized by protocol.

The interrupt trigger by the network adapter is propagated to upper-level protocol components, driving the protocol's state machine and eventually reaching the application whenever a valid message is received. After the components have finished processing the buffer, it is released, causing the allocated memory to return to the system's heap.

To send packets over the network, TSTP allocates a buffer providing information about all headers and the payload just like it does for receiving. The Radio component allocates and MAC initializes the buffer. A pointer to the allocated buffer moves around the protocol components, alongside methods to access the data given by the application. Each relevant component makes its contributions to the headers and metadata, the Security component encrypts and signs the message when necessary, and the buffer returns to the MAC for injection in the network.

Code Example 3.1 illustrates the internal process of allocating a zero-copy buffer, creating a `Keep Alive` message on the memory provided, passing the buffer throughout the components for marshaling, and finally inserting it on the MAC's transmission queue.

3.1.2 Metadata

The zero-copy buffer parameterized class described above takes both the protocol and the target network as parameters. In this way, buffers can be extended with additional, protocol-specific *metadata*. Instead of sharing their data structures to exchange collected knowledge

medium is the case for most Ethernet adapters on Intel and many ARM platforms, but it is not used in TSTP for IEEE 802.15.4 to preserve as much memory as possible free for the application.

```
void TSTP::keep_alive() {
    Buffer * buf = radio->alloc(sizeof(TSTP::Keep_Alive));
    new (buf->frame()) TSTP::Keep_Alive;
    marshal(buf);
    radio->send(buf);
}

void TSTP::marshal(Buffer * buf) {
    locator->marshal(buf);
    timekeeper->marshal(buf);
    router->marshal(buf);
    security->marshal(buf);
}
```

Code Example 3.1: TSTP message creation example.

about the network, components use the metadata in the buffer for interaction with each other. In TSTP, the metadata is not transmitted over the network along with messages. When a message is received at a node, each component that receives the corresponding buffer populates and adjusts specific portions of the metadata contained in that buffer. Similarly, messages coming from the application are encapsulated in a buffer that traverses the protocol from component to component. Both directions follow a pre-defined ordering as shown in Figure 1. The main metadata currently used in TSTP are shown in Table 1.

Besides metadata, TSTP buffers carry 9 bytes of control information: a one-byte lock, a four-byte owner pointer, and a four-byte message size counter. Each buffer also carries 127 bytes for the IEEE 802.15.4 Protocol Data Unit (PDU), which might be transmitted through the network. The common TSTP header, present in the PDU, is shown in Section 3.2. Methods to access and interpret the enclosed message are also provided. Buffers specialized to other protocols export a link to other fragments of the same message. As TSTP buffers do not need nor support fragmentation, this is not present in the TSTP-specialized class, causing no memory overhead.

3.1.3 Event Propagation

Events are propagated through the protocol in two different flows: from application to the network, and from network to application. On the first path, from application to the network, events are

Name	**Set by**	**Used by**	**Meaning**
RSSI	MAC	Locator	Signal reception strength for distance estimation
SFD	MAC	Timekeeper	Time of reception for clock synchronization
ID	MAC	MAC	Message ID to detect retransmissions
Offset	Router, MAC	MAC	Contention offset for channel access
Destined to me	Locator	Router, MAC, API	Whether this message is destined to this node
Direction	Router	Any	Whether this message goes towards the sink
Distance	Locator	Any	This node's distance to the destination
Sender distance	Locator	Router	Last hop's distance to the destination
Trusted	Security	API, MAC	Whether the message's authenticity was verified
Relevant	Any	MAC	Signal the MAC that it must receive the message
Transmissions	MAC	MAC	Incremented each time the MAC transmits this buffer

Table 1: Main buffer metadata used by TSTP.

implicitly propagated as components are invited to visit a buffer in a predefined order through the invocation of specific methods that collaboratively marshal a valid network packet. On the counterflow, from the network up to the application, events are propagated using the Observer design pattern (GAMMA et al., 1995). When the network adapter finishes copying a packet into a previously allocated buffer and triggers an interrupt, that interrupt is handled by the Radio mediator and converted into a *notification* of an observed object (the Radio). TSTP protocol components are therefore observers of zero-copy buffers, as illustrated in Figure 13.

Figures 15 and 16 show the sequence diagrams for sending and receiving messages. Speculative space-time synchronization happens during the execution of the Timekeeper's and Locator's `update` methods. The `send` method of the API ends by inserting the message in the transmission schedule, which is later consulted by the MAC. As the diagrams denote, pointers to buffers are requested to the radio with the `alloc` method, and only these pointers are passed through components, eliminating any copies of memory chunks.

Some network interface controllers feature a relatively large pool of hardware-accessible memory which is mapped into memory directly accessible by software. In these cases, the network hardware mediator might return chunks of this memory as a result of calls to `alloc`, such that messages are constructed in-place, never leaving memory areas that the hardware uses to send messages through the network. Some other devices, such as EPOSMote III, export a very limited amount of such memory, so that the radio mediator returns chunks of memory from a pre-determined region of system memory, and copies it to radio-accessible memory only when the message is actually transmitted. It is the responsibility of the radio mediator to manage this hardware-accessible memory. Whenever the radio mediator runs out of buffer memory, `alloc` returns 0.

3.1.4 Active Components

The main TSTP components are passive, i.e., do not execute in a dedicated thread and are triggered either by hardware interrupts or other active components, such as the application.

The MAC does not run in a dedicated thread, but implements a state machine triggered by a dedicated hardware timer (Section 3.5). Figure 15 does not end with an actual message transmission, but rather an `enqueue` operation on the MAC's transmission schedule. This schedule is consulted by the MAC's state machine when appropriate to actually transmit the message. The MAC does not execute in a dedicated thread because it needs to observe very precise timing (Section 3.5), and scheduling overhead could potentially compromise its correct functioning.

Figure 16 shows that the interrupt handler for received radio messages reaches all the TSTP components, with a decoupling only when the Application needs to be notified. This design decision is also motivated by performance concerns, and components' `update` methods can't perform time-consuming computations. When such computations are needed, they should be decoupled from the critical path of message processing. For example, for the more expensive key establishment cryptographic operations, during system initialization the

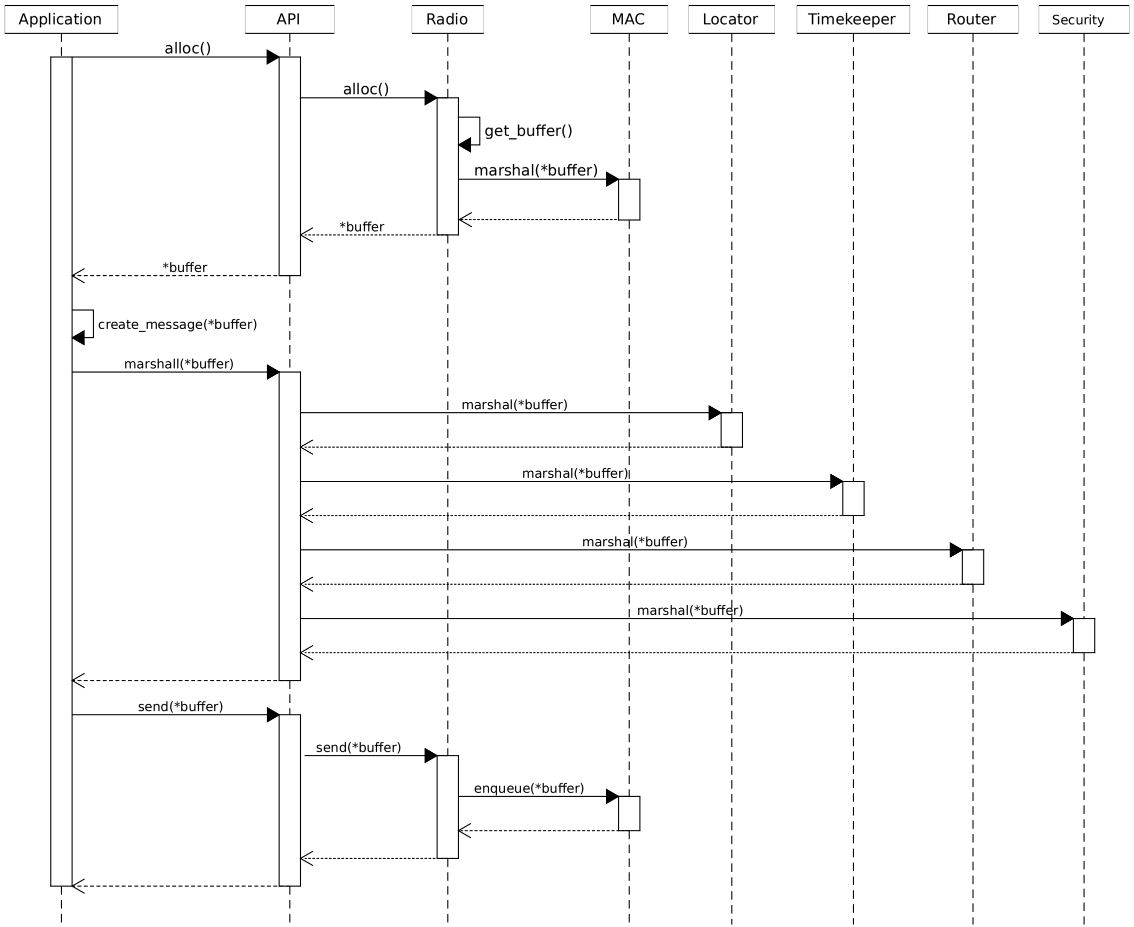


Figure 15: Sequence diagram for message allocation and transmission.

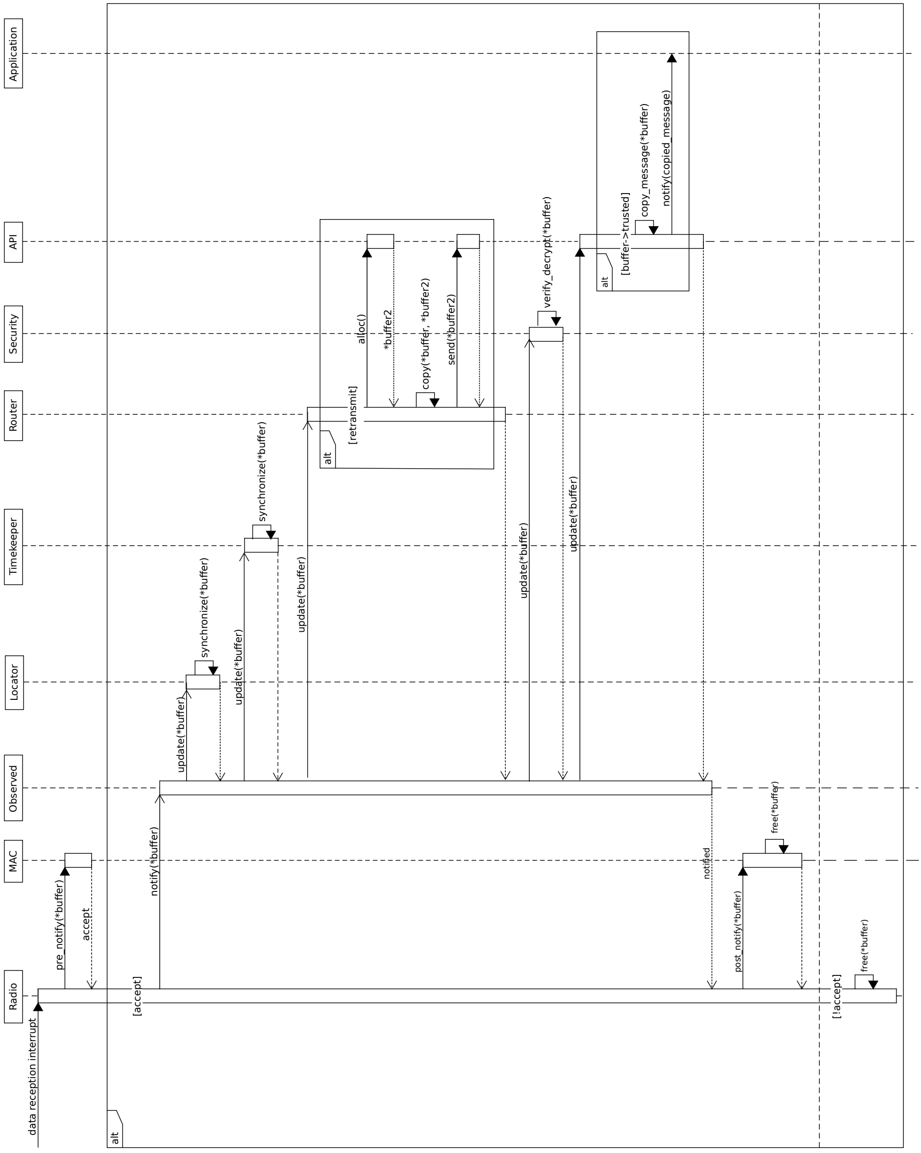


Figure 16: Sequence diagram for message reception and processing.

Security component creates a `Key Manager` active component (a periodic system thread) to manage cryptographic keys.

Besides the key manager, the other active component instantiated by TSTP is a `Lifekeeper`. It is a periodic thread that triggers the execution of TSTP's `keep_alive` method (Code Example 3.1). At every execution of the Timekeeper's `update` method in which it is able to synchronize in time, it resets the timer for this thread, so that it only triggers `Keep Alive` messages when a pre-defined period has passed with no messages on the network (Section 3.4).

3.1.5 Bootstrapping

To be able to function in the network, TSTP devices must be synchronized in space and time, and authenticated with the sink. A device with no notion of space can't route messages. A device with no common notion of time can't successfully run the authentication algorithms. Each TSTP component implements a `bootstrap` method, which is called in the same order as when marshaling a new buffer: Locator, Timekeeper, Router, Security, API. The `bootstrap` method is invoked once during initialization of the system, and each component should register as a Radio observer and block until it is successfully synchronized. This means that Timekeeper only runs after the device knows where it is, Security only receives messages after the device is synchronized in time, and the application will only start receiving data after all the components are initialized[4].

3.1.6 Interaction with the API Component

The API component is realized as a singleton class named TSTP, containing pointers to all the other components. Applications can ask the protocol for the node's current location with the `TSTP::here()` method, which will be redirected to the Locator. Likewise, the current time in microseconds is consulted with `TSTP::now()`, and is redirected to the Timekeeper. TSTP exports to applications the `Interested` and `Responsive` classes to represent Interests and Response capabilities. Higher-level APIs such as SmartData leverage these entry-points to

[4] In fact, since TSTP devices can't function properly without authentication and a notion of Space-Time, in the EPOS implementation the application's `main` method won't even be reached before TSTP is initialized.

Header Format

Bits:	**3**	**2**	**1**	**2**	**8**	$\mathbf{3sb + 64}$	$\mathbf{3sb + 64}$	**0 or 32**	
	Version	Type	Time Request	Scale	Location Confidence	Origin (x,y,z,t)	Last Hop (x,y,z,t)	Location Deviation	24 - 46 bytes

Figure 17: TSTP common message Header format.

create powerful and elegant abstractions.

3.2 COMMON MESSAGE FORMATS

The TSTP message header is common to all TSTP messages, and every protocol component can access, parse, and modify any portion of the message. The TSTP header is shown in Figure 17. To enable routing and passive synchronization, the header contains information about the node that is currently transmitting that message: its coordinates and the precise transmission time (`Last Hop` fields), and its confidence in its own estimations of location (`Location Confidence`) and time (`Time Request`).

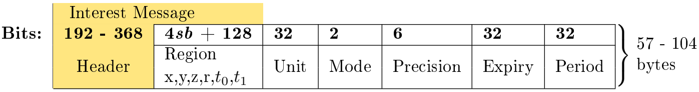
Besides characterizing the last node to transmit a message, the TSTP header characterizes the message itself. The `Origin` fields indicate the time and place where the message was generated. The `Version` field identifies the TSTP version of the message, which is currently always set to 4. This field is packed first and matches the Frame Type field in the Frame Control of IEEE 802.15.4 MAC packets. A version number above 3 renders TSTP into the reserved frame type zone (IEEE..., 2011), so that TSTP packets will not be erroneously interpreted by other coexistent IEEE 802.15.4 nodes running different protocols.

Table 2 shows the meaning of values for the `Scale` field. These bits define the size of each spatial coordinate (denoted sb) in the corresponding message, as well as a multiplier to be applied. For instance, with a `Scale` code 10, each `x`, `y`, and `z` coordinate shall have 16 bits and represent multiples of $25cm$, so a value of $x = 3$ would represent $75cm$. Such scaling makes these fields adaptable to different application scenarios, which may vary from, for instance, monitoring a small room to a large forest.

The possible values for the message `Type` field are shown in Table 3. A sink node announces interest in a physical quantity using `Interest` messages. When receiving an `Interest` message, a sensor node checks if it is inside the desired region and able to measure the

Code	Unit	Scale	Size (bits)	Maximum Value
00	cm	50	8	127.5 m
01	cm	1	16	655.3 m
10	cm	25	16	16382.5 m
11	cm	1	32	42949.6 km

Table 2: Spatial scaling codes.



Interest Message

Bits:	**192 - 368**	***4sb* + 128**	**32**	**2**	**6**	**32**	**32**	
	Header	Region x,y,z,r,t_0,t_1	Unit	Mode	Precision	Expiry	Period	57 - 104 bytes

Figure 18: TSTP Interest message format.

requested quantity with the required precision. If so, the Interest is saved and the node will automatically respond with a `Response` message every `Period` of time until the Interest expires, gets revoked, or the node leaves the interested area.

Code	Type
00	Interest
01	Response
10	Command
11	Control

Table 3: TSTP message types.

The `Interest` message (Figure 18) specifies the space-time `Region` in which the Interest is valid (a sphere in space and a time interval), the desired SI `Unit` encoded according to the SmartData specification (Section 2.7), the periodicity of responses (`Period`), and minimum required `Precision`. Each measurement that is to be triggered by this Interest will have an `Expiry` time associated to it, after which the sink will consider the data invalid. This `Expiry` is part of the SmartData API[5], and is used in TSTP only to detect and drop expired messages while they go through the network. The values of the `Mode` field are shown in Table 4.

`Response` messages (Figure 19) carry the SI `Unit`, detected measurement `Error`, the data `Expiry` as informed by the corresponding In-

[5] In SmartData, applications define expiration times for each piece of SmartData. This value is used to inform scheduling decisions.

Code	Type	Meaning
00	Single	Only one response is desired at each period.
01	All	Responses from all sensor nodes are desired at each period.
10	Revoke	Revoke any interests with matching parameters.
11	*Unused*	

Table 4: TSTP Interest modes.

	Response Message						
Bits:	**192 - 368**	**32**	**8**	**32**	**Variable**	**128**	65 - 119 bytes
	Header	Unit	Error	Expiry	Data	MAC	

Figure 19: TSTP Response message format.

terest, the `Data` itself, and a Message Authentication Code (`MAC`). The SI unit and measurement `Error` are informed by characteristics of the physical sensor that measured the data point. These characteristics are also used to match local sensors to received `Interest` messages' `Precision` and `Unit` fields. When necessary, the Security component pads the data to a size suitable for its cryptographic engine (multiples of 16 bytes for AES), encrypts the data, and signs it with a `MAC`.

`Command` messages (Figure 20) are treated similarly to `Interest` messages, and are issued by the sink to trigger action from actuator nodes. This message type contains the desired `Region` of actuation, the SI `Unit` of the quantity to be acted upon, and a `Value` to apply to it. When directed towards specific sensor nodes, commands are encrypted and signed just as `Response` messages.

`Control` messages are used for various functions such as cryptographic key establishment or reporting of node capabilities, and all carry the common control sub-header shown in Figure 21. The possible `Subtypes` are shown in Table 5. The Control sub-header has plenty of room for introduction of new messages as TSTP evolves and incorporates new features. The first four control message types concern security key establishment, and their corresponding messages are shown in

	Command Message					
Bits:	**192 - 368**	**$4sb$ + 128**	**32**	**Variable**	**128**	80 - 114 bytes
	Header	Region x,y,z,r,t_0,t_1	Unit	Value	MAC	

Figure 20: TSTP Command message format.

	Control Sub-Header	
Bits:	**192 - 368**	**8**
	Header	Subtype

25 - 47 bytes

Figure 21: TSTP Control sub-header format.

	Report Message				
Bits:	**200 - 376**	**32**	**8**	**1**	**7**
	Control Sub-header	Unit	Error	Epoch Request	*Unused*

31 - 53 bytes

Figure 22: TSTP Report message format.

Section 3.6.

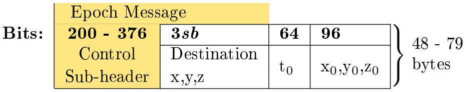
Value	**Subtype**	**Value**	**Subtype**
0	ECDH Request	4	Report
1	ECDH Response	5	Keep Alive
2	Auth Request	6	Epoch
3	Auth Granted	7 - 255	*Unused*

Table 5: TSTP Control subtypes.

`Keep Alive` control messages are "artificial" messages, containing only the Control Sub-header (Figure 21) with `Subtype` 5. Nodes may issue `Keep Alives` when they need to gather data for synchronization, but no message went through its neighborhood for some period of time.

`Report` control messages are sent to the sink by sensor nodes after they synchronize and authenticate. The node will send one such message (exposed in Figure 22) for each transducer it has available, informing the `Unit` and `Error` associated with that device.

Inside the TSTP network, Space-Time is defined in relation to a pre-set reference (usually the sink's coordinates and clock), and the sink can convert the internal coordinate system to absolute Space-Time (coordinates relative to the Earth's center, and Unix time in microseconds) when data are to leave the network. In some cases, sensor nodes need to know the parameters to convert from/to the internal coordinate system (e.g. a node with a GPS receiver needs to convert the GPS global coordinates to sink-centered coordinates; a sensor node with a display showing the date needs to convert local time to Unix time). Sensor nodes with that need will set the `Epoch Request` bit on a `Report Message`, which will trigger a response from the sink in the



	Epoch Message				
Bits:	**200 - 376**	**3*sb***	**64**	**96**	48 - 79 bytes
	Control Sub-header	Destination x,y,z	t_0	x_0,y_0,z_0	

Figure 23: TSTP Epoch message format.

form of an Epoch control message (Figure 23), containing the absolute Space-Time coordinates of the center of the network.

3.3 POSITION ESTIMATION

The Locator component will include the node's current position and location confidence in every message it sends. Non-anchor nodes (with confidence $C_n < 100$) maintain a list of the 3 non-collinear landmarks with the best observed confidence. A landmark is composed of a coordinate with an associated distance estimation and confidence greater than 80 (using estimated coordinates with low confidence leads to a very quick error propagation (REGHELIN; FRÖHLICH, 2006)). Whenever a message is observed in the network, the receiving node checks if the message's confidence value is greater than its least-confident landmark, and if so replaces it, re-calculating its own position and confidence.

A lightweight geometry library was implemented for EPOS, which is used by TSTP to store and parse coordinates, and also to perform 3D trilateration. The 3D Point parameterized class, depicted in Figure 24, takes the coordinate type as parameter, and its only data members are the three coordinates. A Sphere contains a point and a radius. Both classes are compiled with the packed GCC attribute to make sure that the compiler will not add memory padding between data members. This way, coordinates or regions present in TSTP messages (e.g. Figures 17, 18) are directly interpreted in-place as Point or Sphere, with no memory overhead. The Coordinate template parameter is a data type of size according to the Scale used by TSTP, as shown in Table 2.

Variations of HECOPS have been implemented and evaluated using different techniques for two-dimensional spaces (REGHELIN; FRÖHLICH, 2006; PIRES; WANNER; FRÖHLICH, 2008; GRACIOLI et al., 2011), which is generally not a suitable approximation for many TSTP scenarios of interest (e.g. multi-floor buildings). In the context of the present work, 3D localization functions were implemented. The Point class provides functions for performing translations, euclidean distance

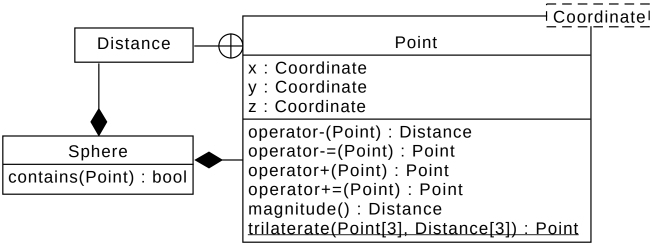


Figure 24: 3D geometry class diagram.

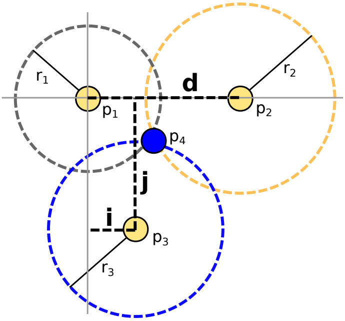


Figure 25: 3D trilateration for finding p_4's coordinates in a coordinate system relative to p_1 and p_2, given three known points and their distances to p_4.

calculation, and trilateration, the latter being the most complex.

To find an unknown 3D point p_4 with trilateration, three known non-collinear coordinates p_1, p_2, p_3 and their respective distances r_1, r_2, r_3 to the unknown point are needed. If $p_1 = (0, 0, 0)$, $p_2 = (d, 0, 0)$, and $p_3 = (i, j, 0)$, as illustrated in Figure 25, then the coordinates x_4, y_4, z_4 of p_4 are determined by the intersection of three circumferences, according to the following equations:

$$x_4 = \frac{r_3^2 - r_1^2 + d^2}{2d} \tag{3.1}$$

$$y_4 = \frac{r_3^2 - r_2^2 + i^2 + j^2}{2j} - \frac{i}{j}x \tag{3.2}$$

$$z_4 = \pm\sqrt{r_3^2 - x_4^2 - y_4^2} \tag{3.3}$$

In the case where the known points are not conveniently located as described, the solution is to build a coordinate system where the conditions are true. To achieve this, four operations must be applied to the three known points: a translation of p_1 to the origin, and three rotations to make $y_2 = 0$ and $z_2 = z_3 = 0$. It is always possible to make one of the coordinates zero for the three points, because three points define a plane. The rotation angles are:

$$\theta_x = \frac{arccos(y_2)}{|p_2|}, \quad \theta_y = \frac{arccos(x_2)}{|p_2|}, \quad \theta_z = \frac{arccos(y_3)}{|p_3|}$$

where $|p_n|$ is the magnitude of point n:

$$|p_n| = \sqrt{x_n^2 + y_n^2 + z_n^2} \tag{3.4}$$

Translation is done simply by subtracting all x coordinates by x_1, y coordinates by y_1, and z coordinates by z_1. Using linear algebra techniques, the two rotations are arranged as a single 3x3 matrix that performs all operations at once when multiplied to a point. The resulting matrix is:

$$\begin{bmatrix} c_y c_z - s_y s_x s_z & -s_z c_x & c_z s_y + s_z s_x c_y \\ s_z c_y + s_y s_x c_z & c_z c_x & s_z s_y - s_x c_z c_y \\ -s_y c_x & s_x & c_x c_y \end{bmatrix}$$

where $c_n = cos(\theta_n)$, and $s_n = sin(\theta_n)$.

After applying the translation and the rotation matrix, Equations 3.1, 3.2, 3.3 can be applied to find the unknown coordinates in the coordinate system based on p_1, p_2, p_3. Afterwards, to return to the original coordinate system, the inverse of the presented rotation matrix must be applied to p_4, and then the initial translation must be reversed by adding the original p_1 coordinates to p_4.

Translations, distances, magnitudes, and operations in the transformed coordinate system are reasonably simple, being adequate for WSN devices. The sine, cosine, and arc-cosine functions can be optimized with lookup tables at the cost of accuracy. The matrix multiplications are the most costly operations for WSN devices. Implementation of a more efficient algorithm for distance-based 3D localization (DOUKHNITCH; SALAMAH; OZEN, 2008) is left as future work.

3.4 TIME SYNCHRONIZATION

The Speculative Precision Time Protocol (SPTP) explores the protocol's cross-layer architecture to speculatively peek through the timestamps and geographic information present in TSTP headers with minimal insertion of explicit messages. A combination of low-level timestamping provided by TSTP's MAC, careful software implementation leveraging the control granted by EPOS, and deterministic radio hardware provides the necessary basis for high-accuracy time synchronization.

3.4.1 Sources of Synchronization Imprecision

In the domain of energy-constrained IoT devices, no compensation mechanism can achieve perfect target frequencies under practical conditions. The IEEE 802.15.4 standard, for example, has a precision requirement for devices of ±40ppm, including temperature and aging variations (MEHTA; PISTER, 2011). Because any two clock sources with differing frequencies will always drift apart, a common measure of time between two independent systems requires a synchronization mechanism. The synchronization process itself is subject to temporal inaccuracies and variations. In the case of wireless IoT devices, these variations arise from the message exchange process itself, which typically includes the following steps:

1. Signal radio to enter transmission mode
2. Read the local timestamp
3. Copy message to the radio with the timestamp
4. Send Start of Frame Delimiter (SFD)
5. SFD is received
6. Receiver's current timestamp is recorded

The fundamental source of inaccuracy at the time of synchronization is the *variation of the time interval* (i.e. *jitter*) from acquiring the sender's timestamp (step 2) to acquiring the receiver's timestamp (step 6). If this time was constant and quantifiable, even if very long, synchronization at the time of reading the timestamp would be perfect. This is however not the case in the majority of scenarios, since there are sources of jitter which may come from protocol design, software/hardware implementations, or physical phenomena. In NTP, for example, the delay is widely variable due to dynamic traffic and congestion in the multiple hops between sender and receiver. In single hop scenarios with a shared medium (such as wireless communications), the delay is variable due to medium access and propagation delays. Finally, even hosts using switched networks with stricter timing guarantees may observe jitter due to variations in software interrupt handling times.

In IEEE 802.15.4, the quality of timestamps (steps 2 and 6) is dictated by the standard, which imposes a minimum clock frequency accuracy requirement. The delay d_{TX} involved in SFD transmission (steps 4 and 5) comes from software and hardware processing, medium access, and signal propagation. Software processing can be measured and made deterministic in many embedded systems. The jitter and delay in medium access can be accounted for if the lower layers of the protocol stack are aware of the time synchronization mechanism: the MAC layer or the hardware itself can time-stamp messages after medium access, very close to the exact moment when it is to be sent through the physical medium. Apart from a deterministic time taken to obtain timestamps right before/after an SFD is sent/received, the elapsed time between transmission and reception of timestamps is determined by delays provenient from signal propagation and processing at the physical layer. The former is negligible in the relatively short distances used in IEEE 802.15.4 links[6]; the latter is determined by the particular hardware specification, but mostly by the well-defined delays present in the standard:

$s_r = 62.5\frac{symbol}{ms}$	IEEE 802.15.4 Symbol Rate
$t_u = \frac{12}{s_r} = 0.192ms$	IEEE 802.15.4 Turnaround Time
$S_{PHR} = 10symbol$	PHY header + preamble size
$t_{PHR} = 0.160ms$	PHY header + preamble time

and a more random component between SFD transmission and reception caused by the process of synchronization between the sender's signal modulator and the receiver's demodulator that takes place during the processing of the PHY preamble. Because this is the only component with inherent and non-negligible – but bounded – jitter, it determines the limit for any software time synchronization strategy. This jitter is experimentally investigated in Section 4.2.3, and found to be in the order of nanoseconds for the target platform.

[6] Radio waves on air travel close to the speed of light.

3.4.2 SPTP Implementation

Clock synchronization is implemented as a collaboration between the MAC and Timekeeper components. When a message is about to be inserted in the network, the MAC temporarily disables CPU interrupts and performs steps 1 to 4 listed above, including the raw reading of the device's timestamp counter in the `Last Hop(t)` field of the TSTP header. Likewise, as soon as an SFD is detected, the MAC is responsible to note the time of reception[7] and attach it to the corresponding buffer's metadata. Timekeeper will take these timestamps to adjust the node's clock according to Equations 2.3, 2.4.

Timekeeper is configured at compile-time with a characterization of the timestamp transmission delay d_{TX}, and the maximum tolerable period P between two clock adjustments, which is derived from the clock synchronization accuracy needed by the application (these parameters are explored in Sections 4.2.3 and 4.4.3). Normally, clock adjustments run in speculative mode, just by the observation of network traffic. Whenever the half-life of clock synchronization is reached (every $P/2$ units of time that pass without any message being observed in the network), Timekeeper will send a `Keep Alive` message to prompt neighbors for timestamps. If P units of time pass with no messages, Timekeeper enters explicit mode, and will set the `Time Request` bit on every message it transmits. Messages with this bit set are not used by other nodes to synchronize, because it declares that the node sending it is not synchronized. In explicit mode, the node still sends `Keep Alives` every period $P/2$. The node goes back to speculative mode once it successfully receives enough messages to adjust its clock.

3.5 MAC AND ROUTING

The MAC component interfaces the network with other TSTP components while implementing a low duty cycle, receiver-based, packetized-preamble-sampling scheme that supports arbitrary metrics for relay selection. Figure 26 shows the activity diagram of TSTP MAC. The `Update Transmission Schedule` activity maintains the transmission queue Q_i (Section 2.5), checking for a pending buffer transmission. During `Contend Channel Check`, the node sleeps for a time defined by the contention offset Equation 2.7 and checks the channel.

[7] Some WSN devices such as EPOSMote III provide hardware support for noting SFD timestamps, and the MAC leverages it when available.

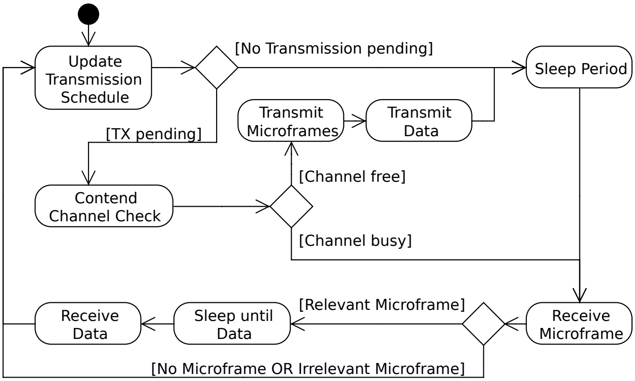


Figure 26: TSTP MAC's activity diagram.

If free, the node proceeds to `Transmit Microframes` for a full period S, with strictly defined timing explained next. It then proceeds to `Transmit Data`, a simple transmission of the queued message. In `Sleep Period`, nodes sleep for the predefined period S, and during `Receive Microframe` they listen to the channel for a time of $2t_s + t_i$ (Figure 27). If a microframe is received, the node decides whether or not the message is relevant. If the message is relevant, the node sleeps until the time the message arrives (the time is derived from the `Count` field in the microframe, the known time between microframes t_i, and the known time t_s to transmit a single microframe). The `Receive Data` activity simply listens until a data message arrives, with a timeout for the case of interference. When the channel is idle, the MAC will alternate between `Update Transmission Schedule`, `Sleep Period`, and `Receive Microframe`. The idle listening duty cycle thus arises from the adjustment of the period S spent in `Sleep Period` as a function of the (bounded) time spent in `Receive Microframe`.

Because TSTP MAC deals with very precise and constrained timing, its states are implemented as interrupt handlers for the Radio and a dedicated timer. Figure 28 shows a Nondeterministic Finite Automaton (NFA) for the MAC, referencing the software functions that implement each activity shown in Figure 26 and the hardware events that cause state transitions. This NFA was also used to check the behavior of the MAC state machine during development, as explained in Section 4.1.

TSTP microframes follow the format depicted in Figure 29. The `All Listen` field requires all listening nodes to receive a message, even

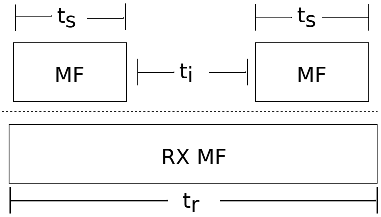


Figure 27: Microframe timing. If $t_r \geq 2t_s + t_i$, it is guaranteed that receiving nodes will have enough listening time to detect and receive at least one microframe.

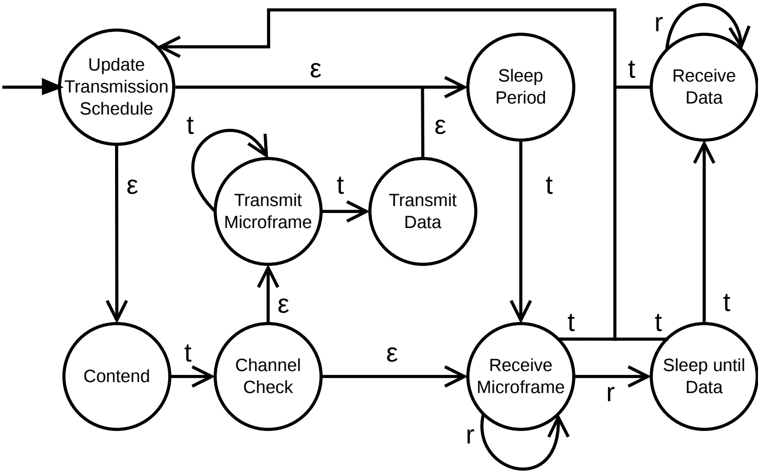


Figure 28: TSTP MAC implementation's automaton. t transitions are triggered by timer interrupts, r denotes radio interrupts, and ϵ represents a transition that happens without a hardware interrupt (e.g. a function call at the end of the previous state's function).

	Microframe Format					
Bits:	**1**	**15**	**8**	**32**	**16**	9 bytes
	All Listen	Id	Count	Distance	CRC	

Figure 29: TSTP microframe format.

those that wouldn't be candidates to relay it because they are further away from the destination. It is used for control messages and for messages whose destination is not a sink and therefore do not follow the default routing algorithm. The `Count` field is initialized by the sending node in accordance with S and decremented for each subsequent microframe until the transmission of the message (after the microframe with `Count` $= 0$). This enables relay candidates to sleep until the time when the message is sent as long as they have listened to at least one microframe. The `Id` field is initialized at the origin (i.e. at the node that effectively produced the data) and accompany the message unmodified across the network until its destination. The purpose of this field is to reduce replication along the route, but not much effort is dedicated to making it unique since the destination has all the information it needs to detect a duplicate in the message's header (i.e. `Origin(x,y,z,t)`). It is currently implemented as a hash of the origin coordinates and time. The `Distance` field holds the calculated distance between the transmitting node and the message's destination, and allows listening nodes to discard a message by listening to a single microframe when `All Listen = 0` and `Distance` is smaller than the listener's distance to the sink. `CRC` is used to discard corrupted microframes.

TSTP MAC takes a toll on senders and on overall latency to achieve low power consumption at receivers. The time t_i between two consecutive microframes is set to a lower bound defined by the underlying communication technology (the IEEE 802.15.4 "turnaround time"), and the time t_s to send a single microframe is fixed and known. To compensate for clock drifts, TSTP MAC defines that the idle listening duration t_r which happens every period S is $2t_s + t_i$ (Figure 27). Therefore, by setting the desired idle listening duty cycle d, the period S can be determined as follows:

$$S = t_s + (N_{MF} - 1) \times (t_s + t_i) \tag{3.5}$$

where N_{MF} is the number of microframes to be sent before each mes-

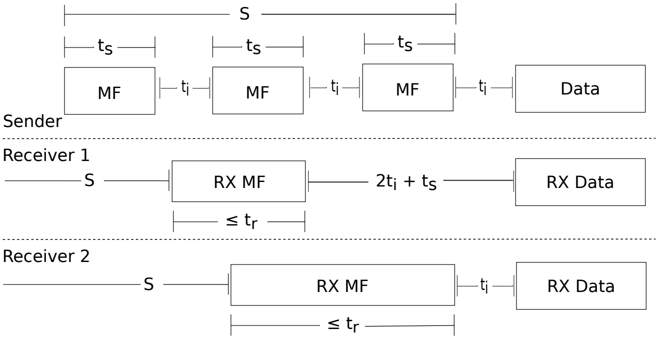


Figure 30: TSTP MAC transmission example with $N_{MF} = 3$. Receiver 2 might wake up late due to clock drift and miss the first Microframe, but it receives the second one if $t_r \geq 2t_s + t_i$.

sage:

$$N_{MF} = 1 + \frac{\frac{2t_s + t_i}{d} + t_i + t_s - 1}{t_i + t_s} \tag{3.6}$$

Figure 30 shows a transmission example with $N_{MF} = 3$. The following list shows the values of these parameters for an IEEE 802.15.4 device.

$s_r = 62.5\frac{symbol}{ms}$	IEEE 802.15.4 Symbol Rate
$T_u = \frac{12}{s_r} = 0.192ms$	IEEE 802.15.4 Turnaround Time
$t_i \geq T_u$	Time between Microframes
$S_{MF} = 30symbol$	Microframe with PHY header size
$t_s = \frac{S_{MF}}{s_r} = 0.48ms$	Time to transmit a Microframe
$t_r \geq 1.152ms$	Microframe listening time

3.5.1 Collisions and Hidden Nodes

Before transmitting any message, a node must sleep for a time offset according to a routing metric (Section 2.5.1) bounded by the MAC period S, then check the channel, and only carry on with transmission in case the channel is detected free. This contention mechanism is appropriate to select relay nodes among contenders and to avoid collisions between neighboring nodes. However, it fails to solve more complex cases of collisions involving non-neighboring nodes. The problem is demonstrated analyzing the traffic captured from a set of

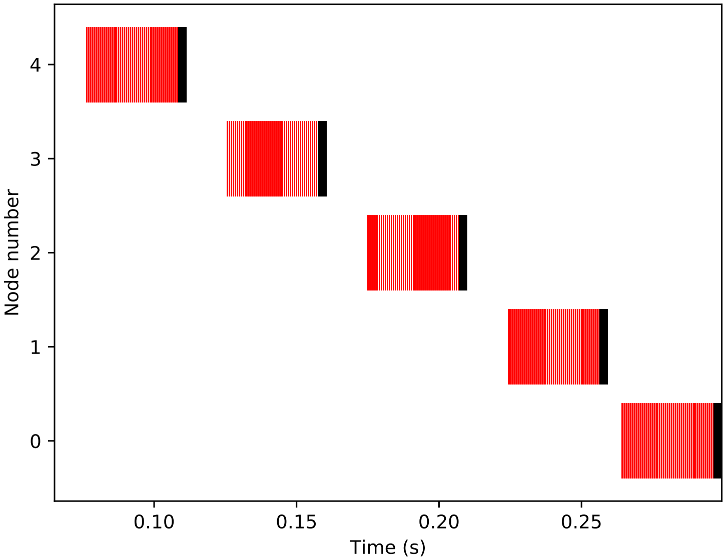


Figure 31: Single message routed from node 4 to node 0. Each node can only reach its immediate neighbors. Red bars represent time spent sending microframes by the node in the corresponding Y coordinate. Black bars represent time spent sending data by the node in the corresponding Y coordinate.

simple simulations.

A scenario with 5 nodes on a line topology (each node has one or two neighbors, with the sink at one extremity) is considered. Figure 31 shows a simple example of a single message being routed from node 4 to node 0 (the sink). The bars on the graph represent TSTP MAC microframes followed by a TSTP message. Bars are colored according to the contents of the first 2 bytes of each message, which in the case of microframes contain the message's ID. Each message has an expiry of 3 seconds, and the MAC preamble is composed of 50 microframes.

Figure 32 illustrates a scenario where the MAC is not able to deliver the messages in time. In this case, every node generates a message at the same time, and nodes are configured to have perfectly synchronized clocks. At the mark of 3s, the messages expire and all nodes stop trying to transmit. This happened because nodes 2 and 0 had a contention offset too close to each other, so that messages always collided at node 1, making it unable to detect the acknowledgment from

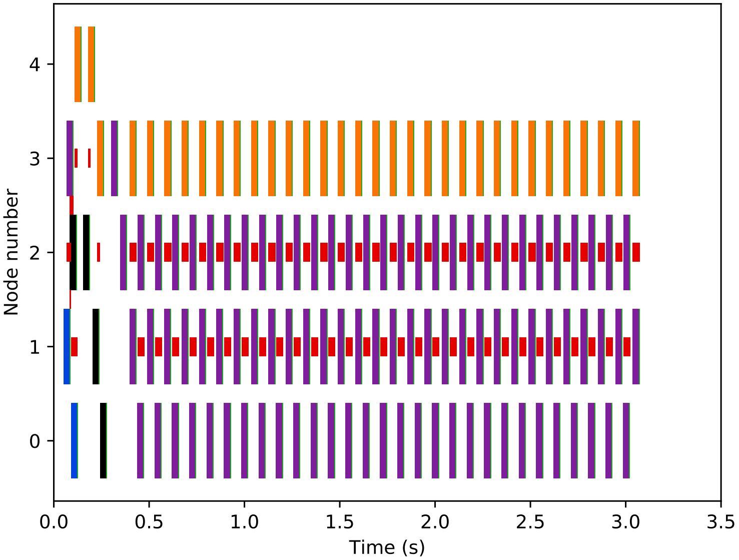


Figure 32: Network traffic with permanent collisions. Different colors represent messages with a different ID. Shorter red bars represent collisions. All sensor nodes generate messages at t=0, but their contention offsets may differ. At t≥0.5, nodes 1 and 2 are hidden nodes. Node 1 transmits the purple message, which is received by the sink (node 0). Then, the sink ACKs it, but node 2 is transmitting at the same time, causing a collision at node 1. Then, node 1 retransmits it for not receiving an ACK, and so on. This continues until the message expires, as contention offsets are not changed.

node 0. Similarly, node 2 is under interference from node 3 and cannot detect that node 1 is forwarding its message.

To solve this kind of permanent collision, a *random backoff* is introduced at the MAC: every time it tries to transmit a buffer, it increments a counter (stored as buffer metadata) and adds to the buffer's offset a random value with upper bound proportional to that counter. In case the message is an ACK, it instead subtracts from the offset. Figure 33 shows the same example with the random backoff employed. There are still collisions, but the backoff slowly pushes node 2 back and lets node 0 acknowledge node 1.

This kind of collision is a classic problem in WSNs, and is known as the hidden node problem. It happens when two nodes detect a clear

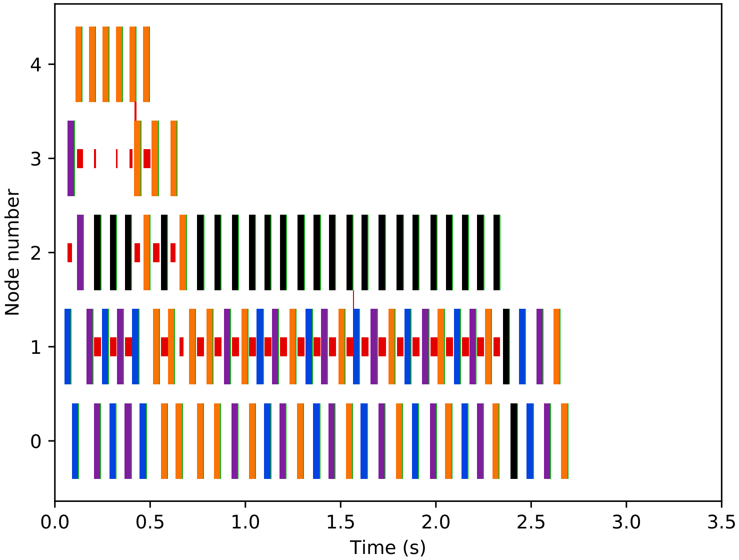


Figure 33: Network traffic with random backoff. After each transmission, a random component is added to the contention offset of non-ACK transmissions, so that hidden node situations eventually end.

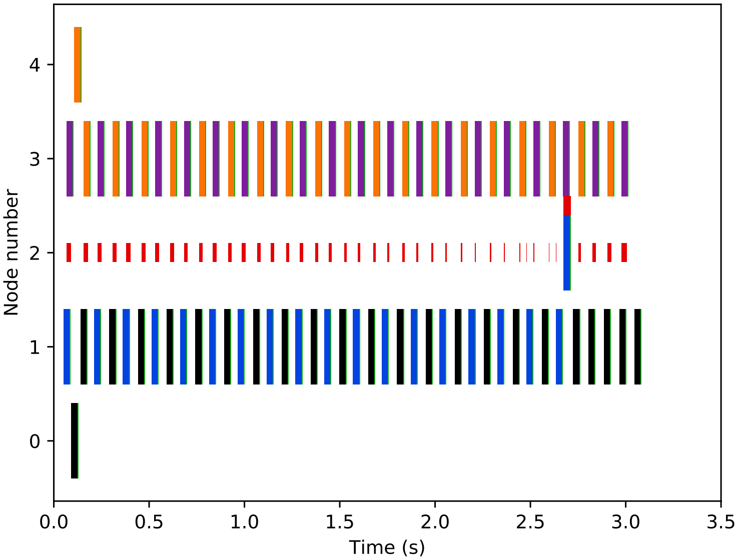


Figure 34: Collisions with sink at the middle with backoff. Nodes 1 and 3 are often occupying the channel at the same time, so that Node 2 (the sink) has a very small chance of correctly receiving a message.

channel for being out of range from each other and start a transmission. However, those messages will collide at a third node within communication range of both transmitters. The random backoff can make the nodes desynchronize their transmissions over time to avoid permanent retries and collisions, but does not completely solve the problem. In Figure 34, the same setup is considered, except that node 2 is the sink. In this case where the sink is a hidden node, random backoff was not enough to deliver the messages in time.

To solve this problem, every node that transmits a non-ACK message will be prevented from transmitting again for a number of MAC periods. This gives receivers of the transmitted message a chance to acknowledge it and forward it out of the hidden node area (POLASTRE; HILL; CULLER, 2004). To implement this, the MAC will keep a counter of how many times any given buffer was (re-)transmitted in the past. Every time it transmits a buffer that is not an explicit acknowledgment, it increments the counter and will not transmit any message for a random number of periods between 1 and the counter value. Figures 35, 36 show that random periods of silence combined with random

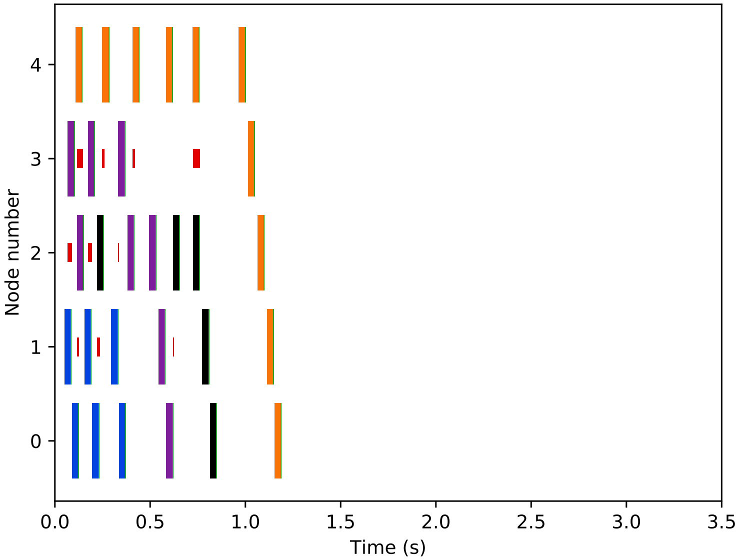


Figure 35: Collisions with sink at the end with backoff and silence. After a transmission, a node will not transmit anything for a randomized number of MAC periods. This gives nodes more time to ACK the messages, and hidden node situations are solved quicker.

backoffs successfully solve the collisions in the observed cases, decreasing total radio activity time and greatly improving end-to-end delay.

To further reduce the cases of interference, a multichannel technique is also introduced. All microframes are sent in a pre-determined, fixed channel, but their corresponding data messages are sent in a channel determined by the message's ID (which is included in the microframe). This way, a node may wake up, receive a microframe correctly, sleep and wake up to receive the data message (in another channel) successfully, even if another node started transmitting its own microframes in the meantime, while that node was sleeping (the other node's microframes would collide with the data message if they were in the same channel).

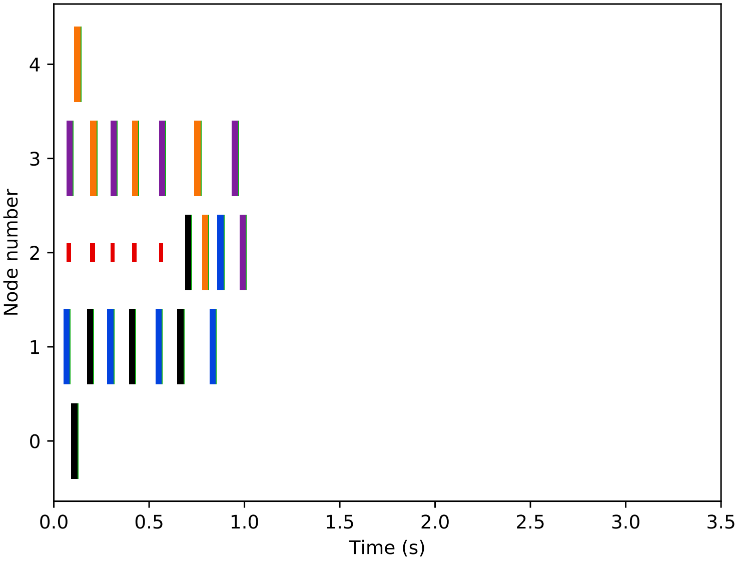


Figure 36: Collisions with sink at the middle with backoff and silence. At around t=0.7, the periods of silence from nodes 1 and 3 are long enough that node 2 has the chance to correctly receive and acknowledge the messages.

3.6 SECURITY

As explained in Section 2.6, TSTP's security mechanisms are based on Elliptic-Curve Diffie Hellman (ECDH) for initial establishment of symmetric keys exchanging two messages: `ECDH Request` and `ECDH Response`. Afterwards, Poly1305-AES is used to combine unique sensor IDs and synchronized time-stamps for generation of One-Time-Passwords (OTP) used to authenticate those keys by exchanging two more messages: `Auth Request` and `Auth Granted`. Cryptographic keys are never used directly, but combined as OTPs which are then used as AES symmetric keys to sign, verify, encrypt, and decrypt messages.

AES is an attractive cryptographic engine because of the relatively common presence of hardware accelerators for it in WSN devices, in part due to it being adopted by the IEEE 802.15.4 standard. AES hardware accelerators are used by the Security component when available (such as in EPOSMote III), and a software implementation is provided otherwise (such as in OMNeT++). AES realizes the `Cipher` interface, which simply exports an `encrypt` and a `decrypt` method that are used by TSTP. The actual AES implementation to be used is defined at compile-time with static metaprogramming techniques, incurring no runtime overhead.

To support the ECDH operations, a Bignum library was developed for EPOS (RESNER, 2014) to implement prime finite field arithmetics with large numbers. It uses the thin template technique to provide an elegant and light interface to interact with arbitrary-sized arrays of "digits", each digit being a native word of the CPU. The Bignum library uses the Barrett Reduction algorithm to implement efficient addition, subtraction, multiplication, and inversion modulo a large prime number. An ECDH library was developed on top of this Bignum implementation, with efficient elliptic-curve point multiplication using jacobian and affine coordinates. TSTP currently works with a key size of 128 bits.

The Poly1305 implementation uses the Bignum library for its operations modulo $2^{130} - 5$, and the `Cipher` for its interaction with AES, being as efficient as those implementations.

3.6.1 Key establishment and management

Although TSTP's security mechanism is currently used to establish end-to-end encryption and authentication between the sink and

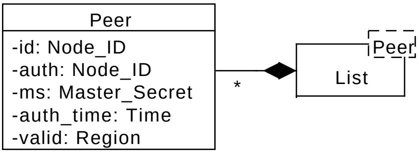


Figure 37: Security component's `Peer` data structure.

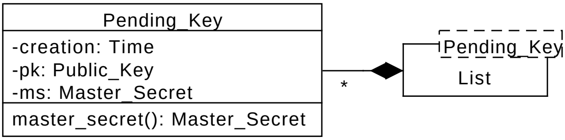


Figure 38: Security component's `Pending_Key` data structure.

each sensor node independently, its design does not prevent sensor nodes from acting as masters, or from establishing keys with multiple masters. The Security component's data structures are implemented in a way to reflect that, naturally supporting expansion to group keys, sensor-to-sensor keys, or multiple sinks, as well as providing a uniform key management interface for masters and slaves.

The Security component maintains two lists of `Peers`, which are called `pending_peers` and `trusted_peers`, and a list of `pending_keys`. Figures 37, 38 show simplified diagrams of these data structures.

At any time, a master can add `Peers` to the `pending_peers` list, representing a slave node that will eventually try to authenticate with this master. It initializes the data structure with the ID and Auth of the slave, and a space-time region in which that node is allowed to authenticate. Upon insertion of the first slave, a master will start a key management thread, which will periodically examine both lists of peers to remove expired entries and send `ECDH Request` messages (Figure 39) to valid pending peers[8].

	ECDH Request Message			
Bits:	**200 - 376**	**3*sb***	**384**	76 - 107 bytes
	Control Sub-header	Destination x,y,z	Public Key	

Figure 39: TSTP ECDH Request message format.

At bootstrap time, a slave will instantiate a `Peer` containing its

[8] The study of appropriate key manager periods and key expiration times are not in the scope of this work.

own ID and Auth, and a space-time region centered at the coordinates of the master, radius 0, and time equal to the expected lifetime of the node.

Upon reception of an ECDH public key (either a slave receiving an `ECDH Request` message or a master receiving an `ECDH Response` – Figure 40), a node will check if there are any pending peers, and if so, save the public key and the current time in a `Pending_Key` object and insert it in the `pending_keys` list. The public key is only used to calculate the Master Secret, but the `Pending_Key` object saves it to use lazy evaluation, calculating the Master Secret only the first time it is actually accessed (by means of the `master_secret` method). This prevents the costly ECDH point multiplication in cases where the upcoming authentication step fails.

	ECDH Response Message		
Bits:	**200 - 376**	**384**	73 - 95 bytes
	Control Sub-header	Public Key	

Figure 40: TSTP ECDH Response message format.

Upon reception of an OTP (either a master receiving an `Auth Request` – Figure 41 – or a slave receiving an `Auth Granted` message – Figure 42), the node tries each possible pending key/pending peer pair to generate a matching OTP. Once it does, the Master Secret is copied to the `Peer`, its authentication time is registered, that `Peer` moves to the `trusted_peers` list, and the `Pending_Key` is removed from the `pending_keys` list and destroyed.

	Auth Request Message			
Bits:	**200 - 376**	**128**	**128**	57 - 79 bytes
	Control Sub-header	Auth	OTP	

Figure 41: TSTP Auth Request message format.

	Auth Granted Message			
Bits:	**200 - 376**	**3sb**	**128**	44 - 75 bytes
	Control Sub-header	Destination x,y,z	Auth OK	

Figure 42: TSTP Auth Granted message format.

The `trusted_peers` list is checked by the Security component to find the necessary shared secrets to encrypt, decrypt, sign, and verify each message.

3.7 SMARTDATA

The SmartData API is out of scope for the present work. SmartData is seen as “application” by TSTP, and none of the evaluations in this work consider it. From TSTP’s perspective, applications can interact with it using a full implementation of SmartData – such as in the EPOS implementation, – or a minimalist compatibility layer – such as in the OMNeT++ implementation.

4 TSTP EVALUATION

Parts of this chapter appeared earlier in:

- *Speculative Precision Time Protocol: submicrosecond clock synchronization for the IoT*
  (RESNER; FRÖHLICH; WANNER, 2016)
- *Design and Implementation of a Cross-Layer IoT Protocol*
  (RESNER; ARAUJO; FRÖHLICH, 2017)

The wide range and complexity of TSTP requires a performance evaluation using different approaches and tools. This chapter tackles validation and performance evaluation from different angles: an implementation for the EPOSMote III platform allows validation of the protocol in real-world deployments, as well as measurements that are heavily influenced by environment or hardware conditions (such as time synchronization); a simulator implementation allows debugging in a controlled environment, as well as analysis of arbitrary statistics from a wider range of deployments and parameter combinations; an analytic model of network behavior is developed and its predictions are compared against simulation results; code overhead is analyzed for both the simulator and real-world implementations; trace visualization, code profiling, and other tools are used to help debug and optimize implementations.

4.1 TOOLS AND DEBUGGING

The distributed nature of WSNs, combined with the hardware constraints of the devices involved, results in a development environment that can be challenging to test and debug. A series of tools were used and developed to help this process. This section briefly presents these tools, as well as the steps taken to make simulations run faster.

4.1.1 MAC State Machine Verification

To verify the state transitions of the MAC, the code was instrumented during development to indicate, by printing specific characters on the screen, whenever a MAC timer or radio hardware interrupt happens, and whenever a MAC state is entered (Section 3.5). These execution traces are processed by a separate C++ program which implements an NFA with the same structure as the MAC's (Figure 28) and checks whether any state violation is found. This process is specially important for the EPOS implementation because EPOS allows Interrupt Service Routines (ISR) to be preempted. ISR preemption becomes more likely to occur as the MAC deals with events that happen close to each other in time. The check of execution traces produces evidence that there is no unintended preemption of handlers that leads to inconsistent state transitions.

4.1.2 Security Library Verification

For verifying the big integer library used in security (Section 3.6), an EPOS application was developed that generates random numbers and operations, operates them, and prints the numbers, operation, and result with Python list syntax. Then a Python program (which naturally supports big integer arithmetics) parses the outputs, reproduces the operation, and checks the results, reporting any errors at the end. A sample EPOS output is shown below.

```
Bignum Utility Test
 sizeof(Bignum<16>) = 16 bytes.
 sizeof(Bignum<16>::Digit) = 4 bytes.
Random seed = 12345
Modulo = [4294967294, 4294967295, 4294967295, 4294967293] +1
a = [229283573, 3596950572, 2802067423, 3554416254]
b = [2941955441, 3441282840, 1051550459, 3256818826]
a + b = [3171239015, 2743266116, 3853617883, 2516267786]
a = [3490719589, 1157490780, 3144468175, 3866696494]
b = [2103497953, 1538207304, 1511588075, 2684337210]
a * b = [180798668, 400923757, 2054889077, 1345744013]
```

The Elliptic Curve Diffie-Hellman library is tested with an application that generates many random public-private key pairs, operates them, and checks whether the resulting shared keys match. Poly1305-AES signatures are tested against known vectors (BERNSTEIN, 2005), as well as rounds of signing random messages and verifying the signature, then verifying with wrong parameters.

4.1.3 Network Traffic Visualization

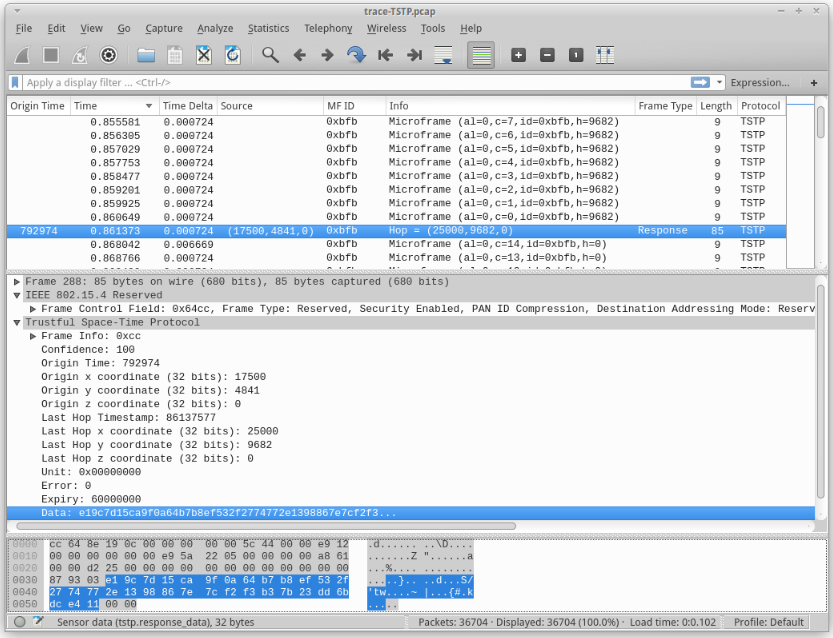
Network traces are useful to verify message formats, timings, and general network behavior. The packet capture (pcap) file format provides a popular and simple syntax for storing network traffic traces, which can be parsed by high-level visualization tools such as Wireshark. A simple C++ pcap formatting library was written for EPOS such that a sniffer node can capture traffic without any interference on the observed network. The library was also included in the OMNeT++ code and can be used in the simulations to capture all the generated traffic.

A Wireshark plugin was written to enable it to parse TSTP traffic. The pcap header indicates that traffic type is IEEE 802.15.4, so that Wireshark loads the appropriate dissector. The IEEE 802.15.4 dissector was modified to identify any 9-byte packet as a TSTP microframe (9 bytes is not an allowed IEEE 802.15.4 frame size), and any packet with size greater than 9 and a reserved `Frame Type` as a TSTP message. In these cases, it calls the TSTP dissector for further processing. Figure 43 shows a sample Wireshark session.

4.1.4 Simulation Execution

Thousands of simulation experiments were performed in the context of the present work, many of which took hours to complete, justifying efforts towards code optimization and efficient usage of computing resources. As simulations are numerous, independent, and CPU-bound, a suite of scripts was developed to execute them in parallel, each as a separate Linux process. The scripts deploy the simulations to a 36-core Amazon AWS virtual machine, spawning 36 Linux processes and reporting 100% usage on all CPUs. Simulation runs are randomly assigned to each CPU to avoid particularly long combinations of runs.

Parallelism is not explored within simulation runs, as it would be a more complex task with risks of introducing errors. Instead, to optimize individual runs, a sample simulation was profiled using Valgrind's `callgrind` tool, with `kcachegrind` for visualization. `kcachegrind` showed that a significant portion of time was spent in Castalia's trace generation methods, even when traces were disabled. A pre-compiler macro was added to allow a more aggressive compile-time removal of these methods when they are not desired, and run times were significantly reduced.


trace-TSTP.pcap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Expression...
+

Origin Time	Time	Time Delta	Source	MF ID	Info	Frame Type	Length	Protocol
	0.855581	0.000724		0xbfb	Microframe (al=0,c=7,id=0xbfb,h=9682)		9	TSTP
	0.856305	0.000724		0xbfb	Microframe (al=0,c=6,id=0xbfb,h=9682)		9	TSTP
	0.857029	0.000724		0xbfb	Microframe (al=0,c=5,id=0xbfb,h=9682)		9	TSTP
	0.857753	0.000724		0xbfb	Microframe (al=0,c=4,id=0xbfb,h=9682)		9	TSTP
	0.858477	0.000724		0xbfb	Microframe (al=0,c=3,id=0xbfb,h=9682)		9	TSTP
	0.859201	0.000724		0xbfb	Microframe (al=0,c=2,id=0xbfb,h=9682)		9	TSTP
	0.859925	0.000724		0xbfb	Microframe (al=0,c=1,id=0xbfb,h=9682)		9	TSTP
	0.860649	0.000724		0xbfb	Microframe (al=0,c=0,id=0xbfb,h=9682)		9	TSTP
792974	0.861373	0.000724	(17500,4841,0)	0xbfb	Hop = (25000,9682,0)	Response	85	TSTP
	0.868042	0.006669		0xbfb	Microframe (al=0,c=14,id=0xbfb,h=0)		9	TSTP
	0.868766	0.000724		0xbfb	Microframe (al=0,c=13,id=0xbfb,h=0)		9	TSTP

```
▶ Frame 288: 85 bytes on wire (680 bits), 85 bytes captured (680 bits)
▼ IEEE 802.15.4 Reserved
  ▶ Frame Control Field: 0x64cc, Frame Type: Reserved, Security Enabled, PAN ID Compression, Destination Addressing Mode: Reserv
▼ Trustful Space-Time Protocol
  ▶ Frame Info: 0xcc
    Confidence: 100
    Origin Time: 792974
    Origin x coordinate (32 bits): 17500
    Origin y coordinate (32 bits): 4841
    Origin z coordinate (32 bits): 0
    Last Hop Timestamp: 86137577
    Last Hop x coordinate (32 bits): 25000
    Last Hop y coordinate (32 bits): 9682
    Last Hop z coordinate (32 bits): 0
    Unit: 0x00000000
    Error: 0
    Expiry: 60000000
    Data: e19c7d15ca9f0a64b7b8ef532f2774772e1398867e7cf2f3...
```

```
0000  cc 64 8e 19 0c 00 00 00  00 00 5c 44 00 00 e9 12   .d...... ..\D....
0010  00 00 00 00 00 00 e9 5a  22 05 00 00 00 00 a8 61   .......Z "......a
0020  00 00 d2 25 00 00 00 00  00 00 00 00 00 00 00 00   ...%.... ........
0030  87 93 03 e1 9c 7d 15 ca  9f 0a 64 b7 b8 ef 53 2f   .....}.. ..d...S/
0040  27 74 77 2e 13 98 86 7e  7c f2 f3 b3 7b 23 dd 6b   'tw....~ |...{#.k
0050  dc e4 11 00 00                                      .....
```

Sensor data (tstp.response_data), 32 bytes · Packets: 36704 · Displayed: 36704 (100.0%) · Load time: 0:0.102 · Profile: Default

Figure 43: Analyzing a TSTP pcap trace with Wireshark.

4.2 EPOSMOTE III EXPERIMENTS

Two implementations of TSTP are analyzed in this chapter. At first, TSTP was implemented in C++ for the EPOS Operating System and the EPOSMote III platform, an IoT device based on Texas Instrument's CC2538 System-on-Chip with an IEEE 802.15.4 radio and ARM Cortex-M3 processor at 32MHz.

TSTP's component-based implementation and a programming language compatibility allowed it to be ported from EPOS to the OMNeT++ simulator with the Castalia framework without many changes: mostly rewriting EPOS' network interface mediator to handle Castalia's Radio framework, and the TSTP `API` component's notification method to interact with Castalia's application layer[1]. As a result, the simulator framework implements a very detailed model of a real TSTP network, with simulated network devices running code very similar to what is run on real EPOSMote III devices.

Castalia code was compiled on/for an Intel Core i5-2320 CPU, with g++ version 4.8.5-2ubuntu1 14.04.1 on an x86_64 Ubuntu 14.04 Linux Operating System. EPOS code was cross-compiled on the same host machine with g++ version 4.4.4 for the ARM Cortex-M3 processor. By the time of this writing, all the source code is freely available at `https://epos.lisha.ufsc.br`.

4.2.1 Code Size

Table 6 shows the resulting compiled code size for each protocol component. It shows EPOS' and Castalia's implementations of each TSTP component, compared with two third-party implementations of the AODV protocol, a simple implementation of IEEE 802.15.4 unslotted CSMA/CA MAC for Castalia and EPOS, and BypassRouting, a very simple network layer in Castalia that forwards messages directly between the MAC and application layers. The full TSTP implementations comprising all of the components achieve a code size close to the AODV implementations, which only provide routing.

[1] A few other technical changes were necessary, such as the removal of static methods and class attributes, which in Castalia are shared across every node in the simulated network.

Component	Castalia code size	EPOS code size
Timekeeper	1721	3728
Locator	2687	3048
Router	3134	3606
MAC	4873	7644
API	9936	7434
Security	11148	8322
Total	33499	33782
BypassRouting	1196	-
802.15.4 MAC	5534	3546
AODV 1	29567	-
AODV 2	34513	-

Table 6: Code size (bytes) for TSTP components and other protocols.

4.2.2 Buffer Management

As explained in Section 3.1.1, TSTP uses a relatively simple version of EPOS' zero-copy buffer and a system heap to allocate buffers efficiently. Apart from TSTP metadata and the IEEE 802.15.4 PDU, the buffer uses a one-byte lock, a four-byte owner pointer, and a four-byte size counter as control information, summing up to 9 bytes of overhead.

The time overhead for allocation of buffers was measured on two EPOSMote III devices running TSTP, one acting as a sink and the other as a sensor node. The two nodes allocated a total of 260548 buffers, taking on average 2.31μs for each allocation. For the experiments in this section, hardware interrupts were disabled during the execution of the code parts being assessed.

The mechanism for propagating received buffers throughout the components explained in Section 3.1.3 was evaluated on the same EPOSMote III devices. The time a buffer enters and leaves each component was recorded to find the propagation overhead in relation to the total processing time. For 278800 measurements, the average time for each buffer to leave the MAC, be processed by the other components, and return to the MAC (as depicted in Figure 1) was 108.57μs. From this time, on average 12.60μs were spent between the end of a component's update method and the start of the next, accounting for the overhead of the notification mechanism itself, which represented 11.60% of the total buffer processing time. Table 7 summarizes these measurements.

Buffers allocated	260548
Avg. buffer allocation time	2.31μs
Received buffers processed	278800
Avg. buffer processing time	108.57μs
Avg. notify time	12.60μs
Notify time / processing time	11.60%

Table 7: Assessment of time overhead for buffer management.

4.2.2.1 Integrity Control

Since every component has access to the full message data and metadata, it is not easy to guarantee that components will not change pieces of information that other components may use in another way. There is no intra-node integrity control in the current implementation of the zero-copy buffer. Besides added overhead, it is not straightforward to enforce integrity, because one of the design goals for the current implementation is to keep the components free to use and modify any information, as well as make it easier to try new, previously unexpected subcomponents. The only form of control currently implemented is the well-defined order in which the components' methods for handling buffers are called, as illustrated in Figure 1.

4.2.3 Time Synchronization

The following experiments investigate the limits of time synchronization obtainable in the physical platform. As explained in Section 3.4.1, the jitter in timestamp processing defines the limits of clock synchronization for a given implementation, and arises from the elapsed time in the following steps:

1. Signal radio to enter transmission mode
2. Read the local timestamp
3. Copy message to the radio with the timestamp
4. Send Start of Frame Delimiter (SFD)
5. SFD is received
6. Receiver's current timestamp is recorded

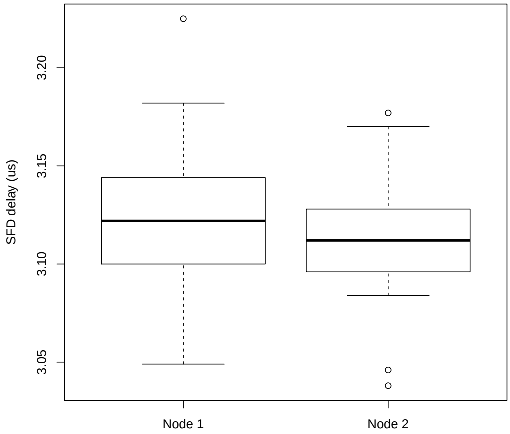


Figure 44: Jitter in Start of Frame Delimiter transmission.

SCHMID; DUTTA; SRIVASTAVA determined the SFD jitter (steps 4 and 5) on the CC2420 platform by measuring the time between the hardware SFD signal (exported by the platform as GPIO) at the sender and at the receiver, and reported that 95% of measurements fell within an interval of $160ns$ (SCHMID; DUTTA; SRIVASTAVA, 2010). For the present work, this experiment was replicated on the EPOSMote III platform. In the setup, one node transmits messages periodically and two other nodes listen. The nodes were positioned close to each other (less than $30cm$ away) with no obstacles in between. The SFD signals (also exported by the platform as GPIO) are observed at each of the three nodes with an oscilloscope and their time differences are noted. The delays observed range from $3.038\mu s$ to $3.225\mu s$, representing an interval of $187ns$ (Figure 44). Using the average value of $3.1315\mu s$, the maximum unpredictable variation of the actual delay for a given transmission is $93.5ns$. This number represents the hardware-imposed limit a synchronization protocol can consistently get at the instant of clock offset determination between two nodes.

Acquiring timestamps close to SFD transmission and reception times requires precise control and predictability in the software stack. In general-purpose systems such as Linux, the wide variability in I/O and interrupt handling timing makes it virtually impossible to accurately quantify the time between these two operations, and hence implementations of protocols such as PTP require hardware timestamping for high-precision synchronization. With hardware timestamping, the network interface itself acquires a timestamp as soon as SFD or equivalent frame delimiters are sent or received. In this work, by disabling interrupts during the short period between message timestamping and transmission and using only deterministic software instructions during this time, the processing delay is accurately compensated. The time from the start of step 1 ("Signal radio to enter transmission mode") to the end of step 4 ("Send Start of Frame Delimiter") was measured with an oscilloscope over several runs of the implementation in question, and it is $352.17\mu s$, $170ns$ above the IEEE 802.15.4 expected time for the radio to get ready and send the physical layer headers $(t_u + t_{PHR})$, with a detected jitter equal to the oscilloscope's own period ($5ns$). The radio hardware itself records timestamps of received messages, and so the delay between SFD reception and timestamp recording (steps 5 and 6) is considered to be zero. Therefore, the combination of low-level timestamping provided by TSTP's MAC, careful software implementation leveraging the control granted by EPOS, and deterministic radio hardware successfully provides the necessary basis for a high-accuracy time synchronization protocol.

To evaluate the performance of the time synchronization algorithm, four motes were set up in a star topology distributed in positions similar to the previous experiment. Three motes are receivers and one is a transmitter, which sends timestamps at a constant interval of 3 seconds (Figure 46), 15 seconds (Figure 47), or 30 minutes (Figure 48). Each receiver node listens to every message and independently synchronizes with the transmitter. When a message arrives at time t_i, the receiver first gets $c_N(t'_i)$ (Section 2.4) without any offset correction, then calculates $\hat{c}_N(t_i)$ as an estimation with its previous calibration variables of what the value of $c_N(t_i)$ is going to be after instant offset determination. It then calculates its immediate offset from the sender (Equation 2.3), updates its clock drift estimation (Equation 2.4), calculates $c_N(t_i)$ taking into account the calibration just performed, and gets the error as $c_N(t_i) - \hat{c}_N(t_i)$. Figure 45 compares the instant offset at the moment of message reception with the previously estimated correction values during execution of Node 1 in Figure 46. The error is the difference between the two. In Figures 46, 47, 48, only the error and instant offset values are shown.

More than achieving a sub-microsecond instant synchronization per-hop close to the physical limit that is comparable to high-precision protocols such as PTP, the results show that on a network with traffic of one message every three seconds, SPTP keeps the network synchronized to sub-microsecond accuracy at *all times* without insertion of any extra

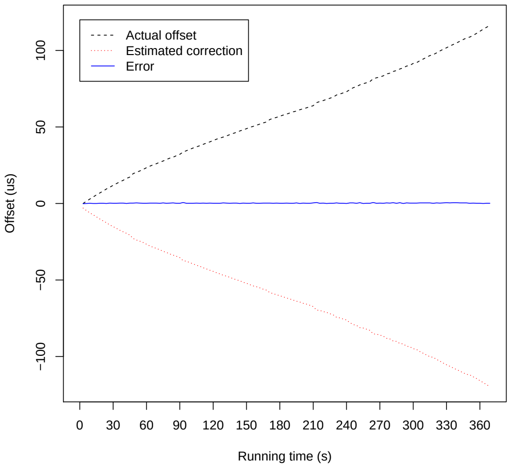
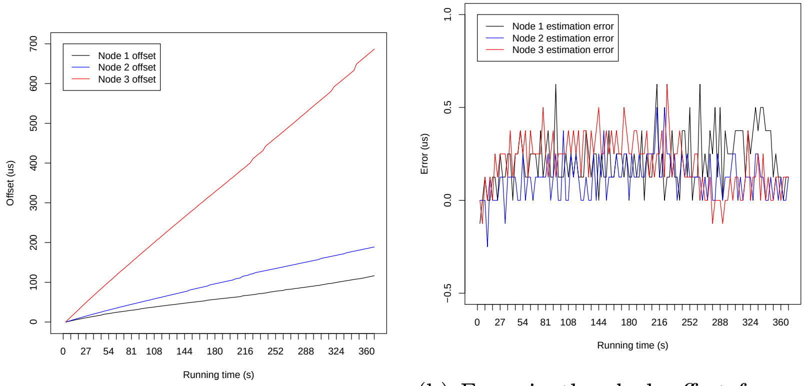


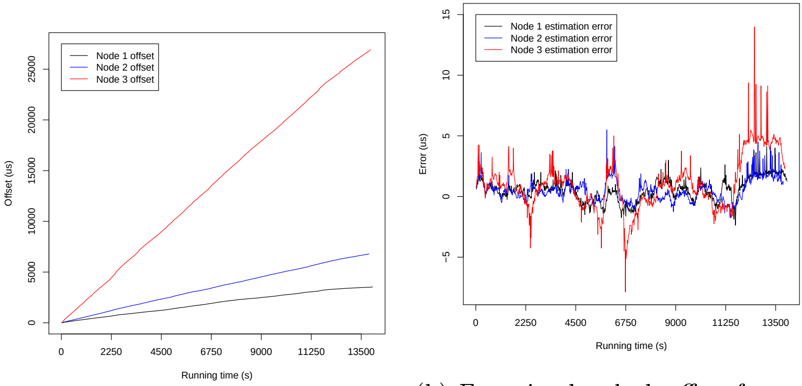
Figure 45: Actual offset from the master before calibration, estimated correction value based on previous observations and its corresponding error for Node 1 in Figure 46.



(a) Clock offsets for each node in relation to the master before calibration.

(b) Error in the clock offset for each node after calibration using a previous estimation of clock drift.

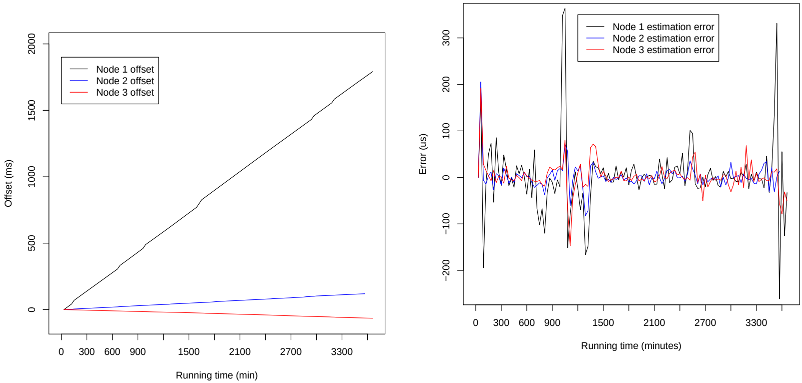
Figure 46: Three EPOSMote III devices synchronizing with a fourth. Messages with timestamp are sent by the synchronizer node every 3 seconds.



(a) Clock offsets for each node in relation to the master before calibration.

(b) Error in the clock offset for each node after calibration using a previous estimation of clock drift.

Figure 47: Three EPOSMote III devices synchronizing with a fourth. Messages with timestamp are sent by the synchronizer node every 15 seconds.



(a) Clock offsets for each node in relation to the master before calibration.

(b) Error in the clock offset for each node after calibration using a previous estimation of clock drift.

Figure 48: Three EPOSMote III devices synchronizing with a fourth. Messages with timestamp are sent by the synchronizer node every 30 minutes.

η	Average number of non-sink neighbors each node has
H	Average number of hops between each node and the sink
λ	Data generation rate of the whole network
S	MAC contention period

Table 8: Analytic model parameters.

message (while PTP, for instance, requires at least 3 explicit synchronization messages). This accuracy gets worse as traffic becomes more sparse, reaching observed worst cases of almost 15μs with a message period of 15 seconds, and around 300μs for 30 minutes.

4.3 ANALYTIC MODEL

The development of an analytic model to predict network behavior under different configurations serves several purposes: it helps understand the way that different factors contribute to different metrics of interest such as average latency or network lifetime; it aids the deployment of new networks, or new sensors on a network, by quickly predicting whether a network with given topology and average data rate will be saturated or close to saturated; it also helps validate the simulator and real-world implementations by comparing their predicted, simulated, or observed behaviors. When used as a network design tool, the analytic model is not meant to be as complex and accurate as the simulator, but much easier and faster to use, providing preliminary insight to designers of new networks.

The whole network is modeled as an M/M/1 queue, which represents a system with a single server, where events arrive according to a Poisson process and job service times have an exponential distribution. Events represent sensor readings that trigger application data messages, and the job service time denotes the total latency between data creation at a sensor node, routing through the network, and acknowledgment by the sink. An M/M/1 queue takes as parameters the arrival rate λ and the mean service rate μ.

The proposed model takes 4 parameters, shown in Table 8. The arrival rate is taken directly as a parameter, and the other three serve to define the mean service rate μ as the average rate at which TSTP can deliver end-to-end messages, considering aspects of topology, collisions, and MAC configuration.

Before sending a message, TSTP MAC determines that a node must pick a time interval called *offset* bounded by the contention period S, wait for that time, start clear channel assessment, and proceed to transmit the message only if no channel activity is detected. The time to send a one-hop message that does not cause collisions is given by Equation 4.1, assuming that the average offset is half the contention period, CCA takes T_{CCA} time, data transmission takes T_d time, and the MAC preamble takes S time.

$$T_1 = (1.5 \times S) + T_{CCA} + T_d \tag{4.1}$$

A *contention slot* is defined as the minimum time difference necessary between two nodes starting CCA so that one of them will detect the subsequent transmission by the other. In IEEE 802.15.4, it is 160μs. Given that contention periods are composed of C_s contention slots, that offsets are chosen at random, and that there are n contending nodes, the probability of any two contenders colliding is determined as the probability of at least two nodes starting CCA at the same contention slot, which is the complement of the probability that every node will pick a different slot, given by Equation 4.2.

$$P_c(n) = 1 - \frac{C_s!}{(C_s - n)! \times C_s^n} \tag{4.2}$$

Because of the MAC silence mechanism (Section 3.5.1), it is approximated that the i^{th} consecutive collision for a given message will cost $(S + T_{CCA}) \times i$ of silencing time, and that the following contention round will have half as many contending nodes. The estimated cost of sending a one-hop message, given that N **other** nodes are contending, and that i consecutive collisions already happened for this message, is thus:

$$\begin{aligned} E_c(N, i) = & P_c(N + 1) \\ & \times (T_1 + (S + T_{CCA})i + E_c(\lfloor N/2 \rfloor, i + 1)) \\ & + (1 - P_c(N + 1)) \times T_1 \end{aligned} \tag{4.3}$$

The mean service rate is defined in Equation 4.4 as the average time a message takes to travel H hops, considering the costs of collisions.

$$\mu = \frac{1}{E_c(\eta, 0) \times H} \tag{4.4}$$

Metrics of interest are then derived from closed-formula values given by the M/M/1 queue model. Average latency is the queue's average service time. Delivery ratio is interpreted as the probability of the service time being higher than the average data generation period, which is given by the exponential distribution with parameter $\mu \times (1 - \lambda/\mu)$. Energy consumption is derived from the probability P_a of the MAC being active, which is the queue's probability of having at least one job being processed in the queue. For network lifetime estimations, it is considered that a given node will be active with probability $P_a/2$, and when active, 75% of the time it will be for receiving a message, and the other 25% for transmitting. The model's predictions are shown in the next section, along with simulation results.

4.3.1 Limitations

The presented model is a simplistic approximation of network behavior in many senses: it assumes that all nodes are homogeneous, with the same behavior and surrounding network traffic, which is generally not the case (e.g. nodes closer to the sink tend to handle more messages); the model of collision overhead and resolution is fairly simple; it completely ignores other relevant effects such as radio signal loss, dropping of expired messages, message duplication along the route, explicit synchronization messages, etc.

The predictions made by the model are only valid when the network is not saturated, since the M/M/1 queue requires that the service rate is higher than the arrival rate.

4.4 SIMULATION EXPERIMENTS

This section presents results from simulations with TSTP's implementation for the OMNeT++ simulator with the Castalia framework. For the remaining of this chapter, the following simulation scenarios are considered: **environment monitoring** and **office**. The environment monitoring scenario models a WSN that monitors slowly changing environmental conditions such as temperature or humidity. It has the following characteristics:

- sparse, constant traffic, with a data period of 60s, 300s, 600s, 900s;
- data expiries equal to period;

- regular placement (Figure 49) on a 500m x 500m field with sink node at the center;
- clean channel: free space model, symmetric radio ranges of ≈ 100m;
- static nodes;
- initial battery charge for every node: two AA batteries (18720J).

The office scenario models an office building with:

- heterogeneous traffic (data period):
  - 10% of nodes at 300ms, expiry = 300ms;
  - 20% of nodes at 1s, expiry = 1s;
  - 60% of nodes at 10s, expiry = 1s;
  - 10% of nodes at 1min, expiry = 300ms;
- two different placements according to real-life office maps: **LISHA** (14 nodes, Figure 58) and **SSB** (40 nodes, Figure 64)
- lossy channel;
- static nodes;
- initial battery charge:
  - two AA batteries (18720J) for 1min nodes and 50% of 10s nodes;
  - infinite battery for remaining nodes.

These characteristics are also presented in Tables 9, 11, 12, when each scenario is analyzed. In all of the scenarios, the sink node has infinite energy supply, and applications start producing periodic data after a random time offset of at most one data period. Each simulation runs for 2 hours (simulated time).

4.4.1 Sources of Random Variation

Each simulation experiment is run with different seeds for the pseudo-random number generators. This seed impacts aspects of the simulation that influence the measured results, even between replicated runs with otherwise identical input parameters. The main sources of random variation in the measured output variables are:

- Each node generates a new data message with a fixed period d. Before sending the first message, each node n waits for a random time $t_n < d$, and proceeds to send a message every d period. Depending on the distribution of these random offsets, nodes may generate messages closer in time, which increases the momentary workload of the network and reduces overall performance;

- The random backoffs and silence periods in the MAC (Section 3.5.1) affect the overall number of message collisions;

- Physical channel effects on the radio signal;

- The clock frequency error for each node is initialized with a random component of at most 40ppm.

4.4.2 MAC Configuration

Correctly configuring MAC and radio parameters for TSTP is not a trivial task. By increasing radio transmission power, nodes will spend more energy to transmit the same amount of data, but radio range will increase and potentially less hops will be needed to reach the final destination, saving energy across the network. Likewise, increasing the length of the MAC preamble causes senders to occupy the channel for a longer time before each data transmission, but allows receivers to sleep for the same, longer, amount of time. There is an optimal configuration for both radio transmission power and MAC preamble length, which is dependent on the network characteristics. The following set of experiments analyze the impact of MAC configurations to identify the optimal ones.

For each scenario, a set of simulations is run to analyze the impact of MAC configurations and select appropriate parameters. For each case, the radio transmission power and the number of microframes in the MAC preamble are varied. The analyzed metrics are delivery ratio, end-to-end latency, and network lifetime.

Radio transmission power is varied between 0dBm and 7dBm. These are the values that are shown in the CC2538 datasheet (TEXAS INSTRUMENTS, 2015) with respective power consumptions of 72mW and 102mW. In the environment monitoring scenario that features a clean channel, 0dBm power results in a radio range of around 60m, and 143m for 7dBm. For the office and industry scenarios, radio ranges are approximately 18m for 0dBm and 35m for 7dBm.

The number of microframes in the preamble define the MAC period and idle listening duty cycle, as defined by Equations 3.5 and 3.6. Longer MAC preambles (with more microframes) should result in an increase of average latency as transmissions take longer to complete, and a decrease in idle listening energy consumption, as channel checks occur more sparsely (Section 3.5). As transmissions take longer, however, the energy consumed by transmitters increase, so there may be a point where further increasing the number of microframes will start to increase the overall energy consumption. The number of microframes observes an upper bound on the resulting MAC period S as defined by Equation 4.5, where D_P is the application data period, and N is the number of nodes.

$$S < \frac{D_P}{(N-1) \times 4} \tag{4.5}$$

If every node were one hop away from the sink and transmissions were perfectly coordinated, the MAC would take at least $2S(N - 1)$ time to transmit and acknowledge every message, so a MAC period S that makes this value larger than the data period D_P would not be able to deliver and acknowledge every message, even in a perfect scenario. This time estimation does not include offsets and silence periods, and is observed in practice to be much higher. Therefore, this upper bound is divided by 2, resulting in Equation 4.5.

In every simulation, the Expiry routing metric is applied (explained next, in Section 4.4.4). For the environment monitoring scenarios, the Timekeeper maximum tolerable synchronization period P (Section 3.4.2) is equal to the data period (which triggers `Keep Alive` messages every $P/2$ time without traffic and Time Requests every P time without traffic). For the office scenarios, it is set to 12s (the sum of the data periods weighted by the proportion of nodes with each data period).

For each scenario, first the number of microframes is varied between 5 to 255 (observing the upper bound given by Equation 4.5) in steps of 5, to analyze general behavior, identifying the point where the network starts showing signs of saturation, and a range of the number of microframes where the optimum network lifetime point is. For each configuration, a vertical dashed line shows the identified optimum number of microframes in terms of network lifetime that still reaches the highest delivery ratio. Simulation results are shown alongside analytical predictions (Section 4.3) from the same configurations. Then, another set of simulations investigates the region around the identified

Number of nodes	116
Field size	500m x 500m

TX power (p)	**0dBm**	**7dBm**
Radio range	60m	143m
Avg. # of hops (H)	3.957	1.939
Avg. # of neighbors (η)	5.200	19.721
TX power consumption	72mW	102mW

Data period (d)	**60s**	**300s**	**600s**	**900s**
Data rate (λ)	1.917	0.383	0.192	0.128
Data expiry	60s	300s	600s	900s
Synchronization period	60s	300s	600s	900s

Table 9: Configurations for environment monitoring scenario.

optimum, varying the number of microframes 1 by 1, with 20 replications for each experiment. Finally, for each scenario variation, a table summarizes the results of the best configuration found for number of MAC preamble microframes and transmission power.

4.4.2.1 Environment Monitoring Scenario

The simulation and analytic model (Section 4.3) parameters for the environment monitoring scenario mentioned in Section 4.4 are summarized in Table 9. Figure 49 shows the network deployment. Figures 50, 51, 52, 53 show the simulation and analytic results for delivery ratio, mean latency, and estimated network lifetime (the time that would take until the first node depletes its battery) for each variation on the environment monitoring scenario. The vertical dashed lines show, for each data period, the microframe configuration that leads to the highest lifetime with 100% delivery ratio, as will be shown in Figure 53.

Figure 50 shows that the MAC can handle the data traffic for all data periods greater than 60s. For 60s, the delivery ratio starts to decrease when the preamble gets bigger than 75 microframes. This is much lower than the loose limit of around 190 microframes imposed by Equation 4.5, and this happens because that equation is highly optimistic for large networks, as it does not consider collisions or even routing delays. It also shows that when using 7dBm transmission power, the delivery ratio starts declining sooner than with 0dBm, indicating

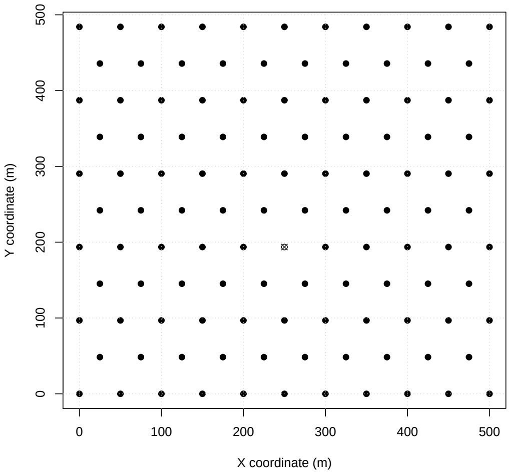
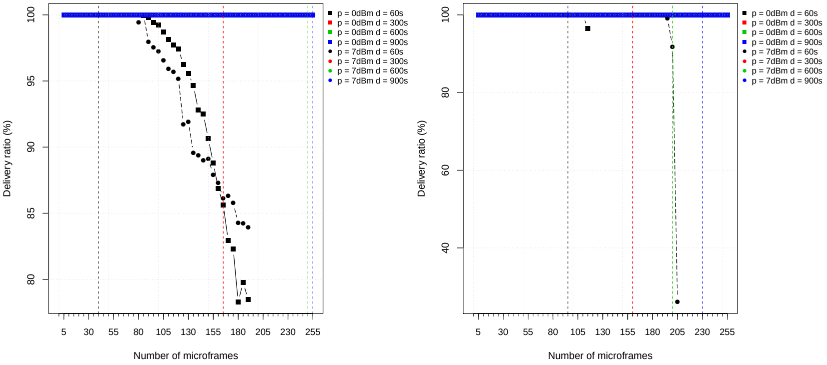


Figure 49: Environment monitoring node map. The sink is close to the middle, marked with an X.

that the longer radio ranges may cause considerably more collisions. These collision effects are not captured by the analytic model, which shows accurate delivery ratio predictions for 0dBm, but is very off for d=60s p=7dBm.

The point where the MAC starts losing messages is also identifiable in Figure 51. For d=60s, the simulations show an abrupt increase in the mean latency at around 90 microframes, showing that a saturation of network capacity escalates quickly, as more messages are being transmitted at any given time and potentially causing collisions. This effect is captured by the analytic model. The figure also shows that the messages that do get delivered in the 7dBm configuration take less time than the ones with 0dBm. Figure 52 shows the same data without the 60s lines. Before MAC capacity is reached, latency increases linearly with the MAC preamble size. A preamble too short (5 microframes in Figure 52) can also increase latency, as the MAC preamble is equivalent to the contention time. A contention window too short will cause more neighbor nodes to have an equal offset and collide (Section 3.5).

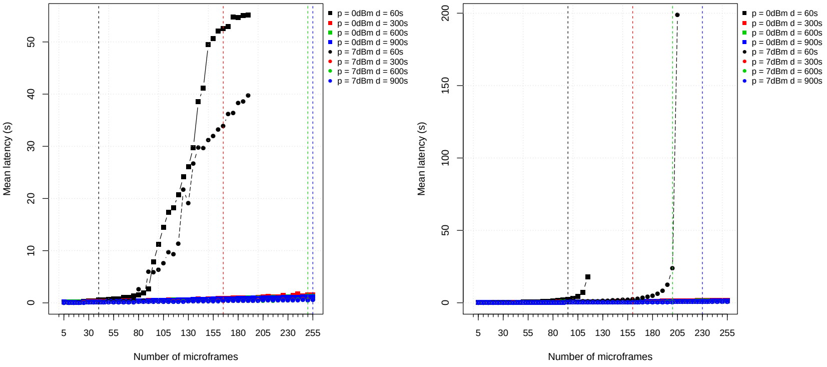
Figure 53 shows the behavior of network lifetime. Although radios transmitting at 7dBm consume more energy than at 0dBm, the 7dBm configurations result in a higher lifetime, because the greater



(a) Simulation

(b) Analytic

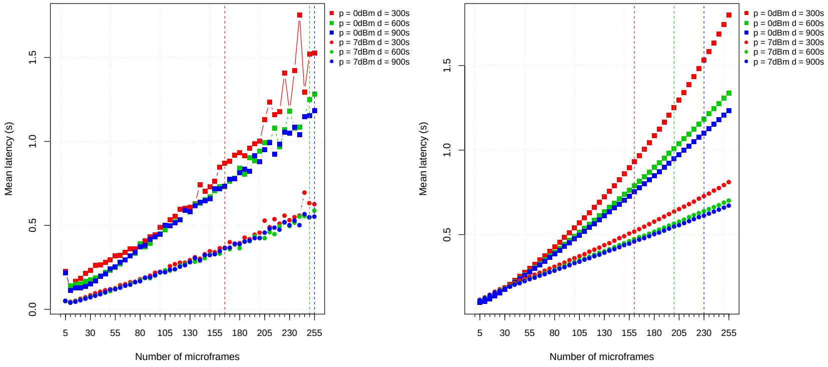
Figure 50: Delivery ratio for environment monitoring scenario. Every line other than d=60s is constant at 100%.



(a) Simulation

(b) Analytic

Figure 51: Mean latency for environment monitoring scenario. Latency grows linearly with the MAC period, determined by the number of microframes, until a saturation point is reached (around 90 microframes for d=60s).



(a) Simulation

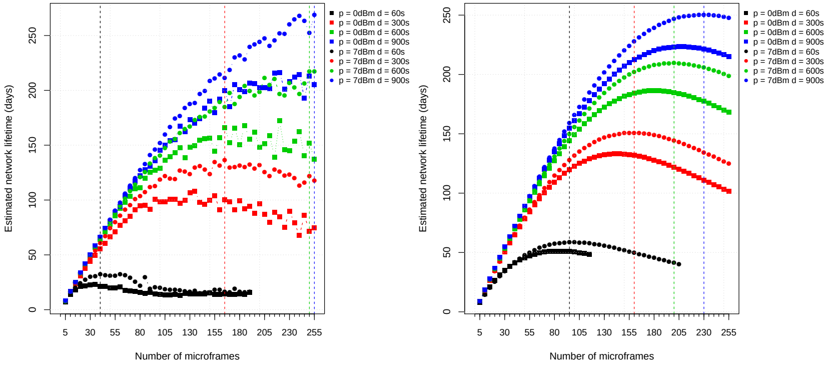
(b) Analytic

Figure 52: Focus on mean latency for environment monitoring scenario with p $>$ 60s. It grows linearly with the MAC period, determined by the number of microframes.

transmission power resulted in a greater radio range, meaning that potentially less hops are necessary to deliver messages, and nodes spend less time with their radios in transmission mode, saving overall energy across the network. More microframes in the preamble means that nodes' idle listening duty cycles are reduced, but it also means that senders spend more energy per message sent. The Figure shows that there is a point where the increase in transmission cost becomes higher than the decrease in receiving costs, and this point varies according to the application data generation period.

Having identified ranges of MAC configurations that produce the best results, more simulations are performed with a finer-grained variation of number of microframes. Figures 54, 55, 56, 57 show these simulation results. For each microframe and data period configuration, 20 replications are run. The shaded areas in these figures represent standard deviations among the means of the 20 replications. In these figures, the number of microframes shows a bigger impact on latency than on lifetime, but that is because the range of microframes considered is one that contains the maximum point in the parabola-like curve that represents lifetime, while latency is expected to grow linearly in that region, as delivery ratio is always 100%.

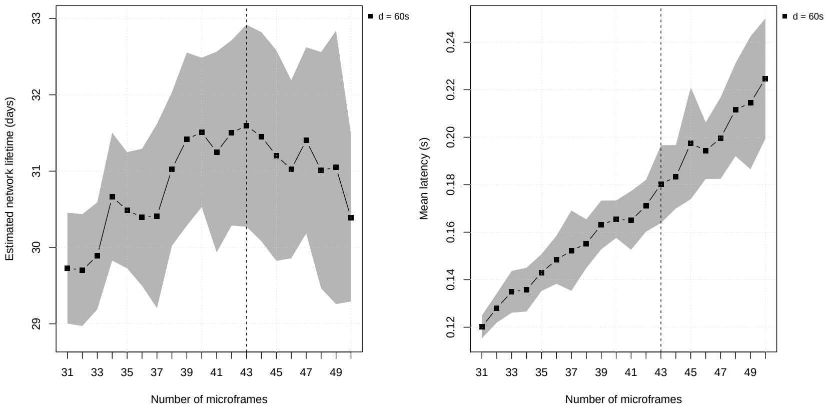
As listed in Section 4.4.1, the sources of variation between simulations with the same parameters that result in the standard deviations shown in the figures are: the initial random time offset of at most one



(a) Simulation

(b) Analytic

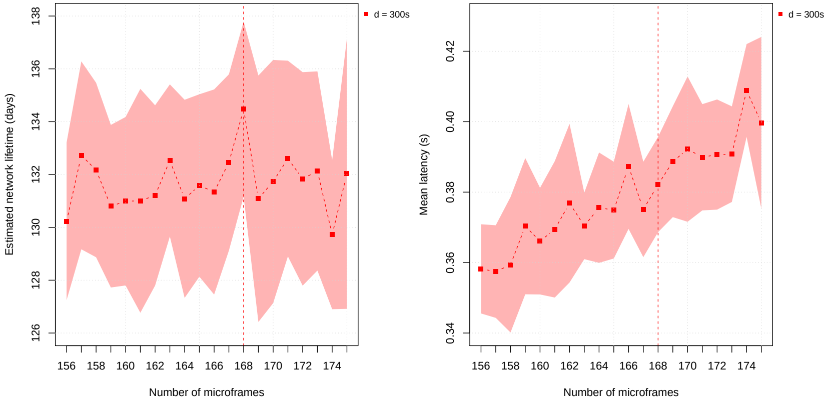
Figure 53: Estimated network lifetime for environment monitoring scenario, for varying transmission power levels (p) and application data period (d). Growing the number of microframes reduces reception cost, but increases transmission cost and time. Lifetime grows proportionally with the number of microframes, until a point where the network starts to spend more energy transmitting than it saves on listening. For each data period, this optimum point is identified with a dashed vertical line. The 7dBm 900s line in the simulation result reached the maximum number of microframes before reaching the optimal point.



(a) Network lifetime (b) Mean latency

Figure 54: Estimated network lifetime and mean latency for selected configurations close to the optimum point of environment monitoring scenario, p=7dBm, d=60s. The optimum network lifetime point is identified as 43 microframes.

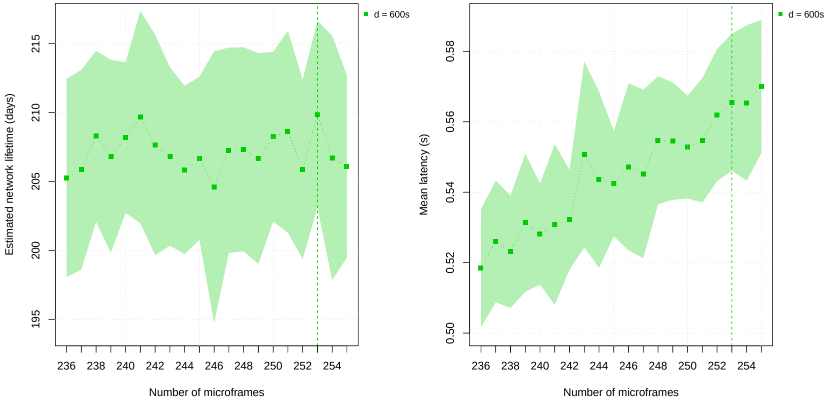
data period with which applications start generating periodic messages; the random backoffs and silence periods in the MAC (Section 3.5.1); channel effects on the radio signal; and the clock frequency error for each node.



(a) Network lifetime

(b) Mean latency

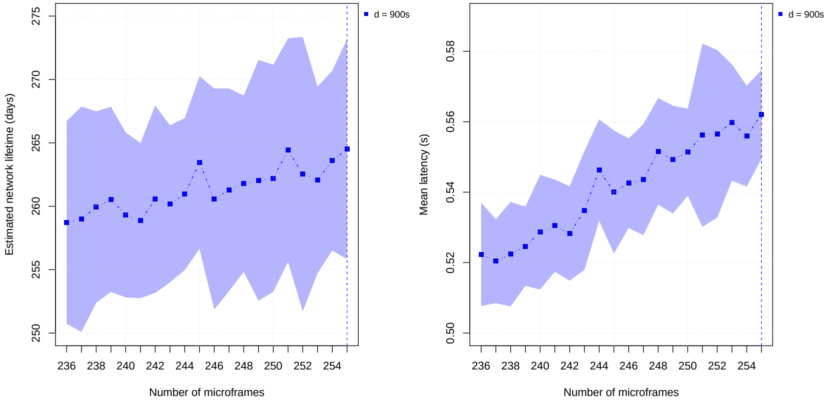
Figure 55: Estimated network lifetime and mean latency for selected configurations close to the optimum point of environment monitoring scenario, p=7dBm, d=300s. The optimum network lifetime point is identified as 168 microframes.



(a) Network lifetime

(b) Mean latency

Figure 56: Estimated network lifetime and mean latency for selected configurations close to the optimum point of environment monitoring scenario, p=7dBm, d=600s. The optimum network lifetime point is identified as 253 microframes.



(a) Network lifetime

(b) Mean latency

Figure 57: Estimated network lifetime and mean latency for selected configurations close to the optimum point of environment monitoring scenario, p=7dBm, d=900s. The optimum network lifetime point would be beyond 255 microframes.

The selected best configurations in terms of network lifetime for the environment monitoring scenario are summarized in Table 10.

Variable	**Unit**				
Data period	s	60	300	600	900
Nodes	-	116	116	116	116
TX power	dBm	7	7	7	7
Microframes	-	43	168	253	255
MAC period	ms	29.51	116.01	174.83	176.21
Delivery ratio	%	100	100	100	100
Network life	days	31.6	134.48	209.88	264.5
Nominal d.c.	%	4.25	1.08	0.72	0.71
Effective d.c.	%	5.54	1.36	0.89	0.83
Max latency	ms	2683.76	2411.74	3015.62	2316.98
Mean latency	ms	180.27	382.16	565.54	562.11

Table 10: Selected configurations for each environment monitoring scenario.

4.4.2.2 LISHA Office Scenario

The two office scenarios model smaller networks than in environment monitoring, but that are subject to a lossy channel, higher data rates and stricter timing requirements. In both variations, the data generation period is fixed across simulations, and nodes are not homogeneous. 60% of the sensor nodes are modeled as nodes that measure non-critical environmental information, generating one message every 10 seconds, with a 10 second expiry. From these, half are battery-powered and half are mains-powered. 20% of the nodes generate data with higher granularity, every 1 second with 1 second expiry, and are mains-powered. 10% of the nodes measure data at a high rate, every 300ms, with equal expiry, and are mains-powered. The 10% remaining nodes represent sensors used for human interaction, sparsely generating data (every 1 minute) with a short expiry (300ms), and are battery-powered. To reach higher lifetimes, the battery-powered nodes do not act as forwarders, they only transmit either data messages generated by themselves, or responses to `Keep Alives` (Section 3.4.2). Table 11 summarizes the configurations for the LISHA office scenario, and Figure 58 shows the node map.

The same experiments shown in Section 4.4.2.1 are performed for

Number of nodes	14
Field size	7m x 5m
Data rate (λ)	8.05
Synchronization period	12s

TX power (p)	**0dBm**	**7dBm**
Radio range	18m	35m
Avg. # of hops (H)	1	1
Avg. # of neighbors (η)	12	12
TX power consumption	72mW	102mW

Node configurations					
Nodes with this config.	1	1	4	4	3
Data period (d)	60s	0.3s	1s	10s	10s
Data expiry	0.3s	0.3s	1s	10s	10s
Battery-powered	Yes	No	No	No	Yes
Acts as forwarder	No	Yes	Yes	Yes	No

Table 11: Configurations for LISHA office scenario.

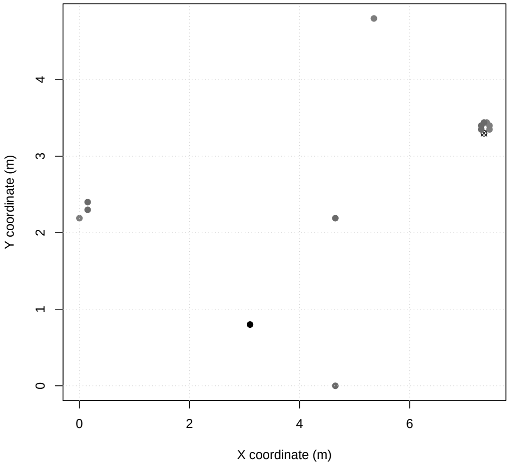
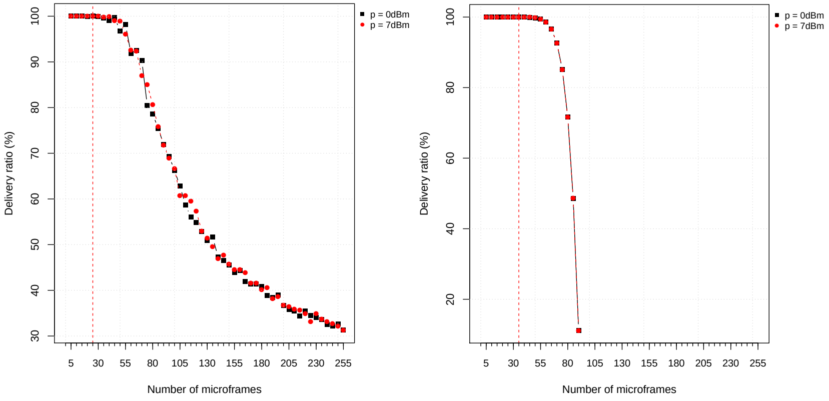


Figure 58: Office LISHA node map. Darker shades indicate higher data periods. The sink is at the rightmost cluster of nodes, marked with an X.



(a) Simulation

(b) Analytic

Figure 59: Delivery ratio for LISHA office scenario. At around 35 microframes, the MAC period gets too large to meet 100% of the expiries.

the two variations on the office scenario. Figure 59 shows that network saturation for the LISHA office scenario starts occurring at around 35 microframes. Moreover, Figure 60 highlights that no configuration is able to deliver 100% of messages all the time. The higher data rate makes it hard for the MAC to honor all the short expiries. However, it is still able to achieve high delivery ratios. Selected configurations (dashed vertical lines) are the ones with the highest network lifetime that present an average delivery ratio minus three standard deviations greater than or equal to 99.99%.

The analytic model still showed optimistic predictions for delivery ratio before network saturation, but was fairly accurate, recommending a configuration of 35 microframes over 25 recommended by the simulation results.

Figures 61, 63b analyze the latency variation, showing that there is not a big difference in mean latency between the two transmission power configurations, as nodes are all at one hop from the destination. The analytic model predicts a latency explosion as the network gets saturated, because it considers homogeneous messages and does not limit the total service time of any message. This subtle increase is not observed in the simulation results, as the 300ms messages expire and are removed, causing less congestion.

Figure 62 shows that the optimal MAC configuration for network life is around 50 to 80 microframes. However, these configurations have

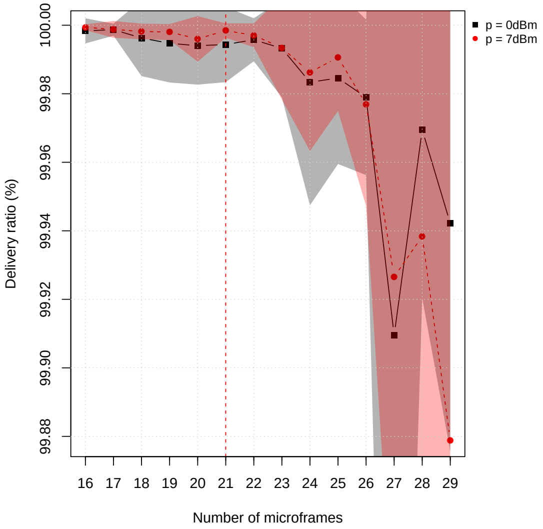
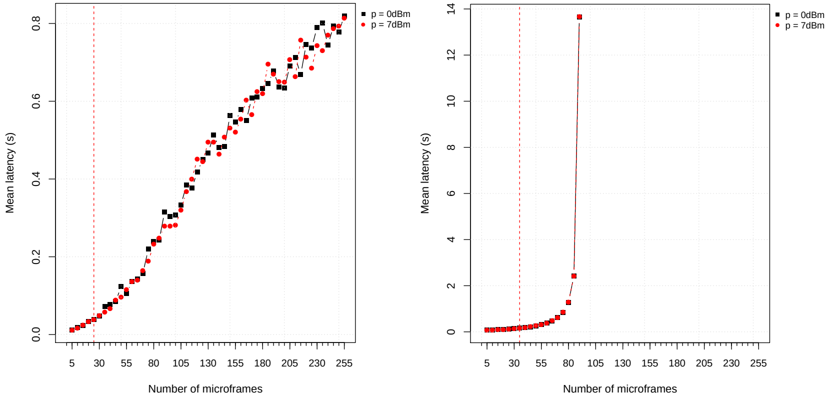


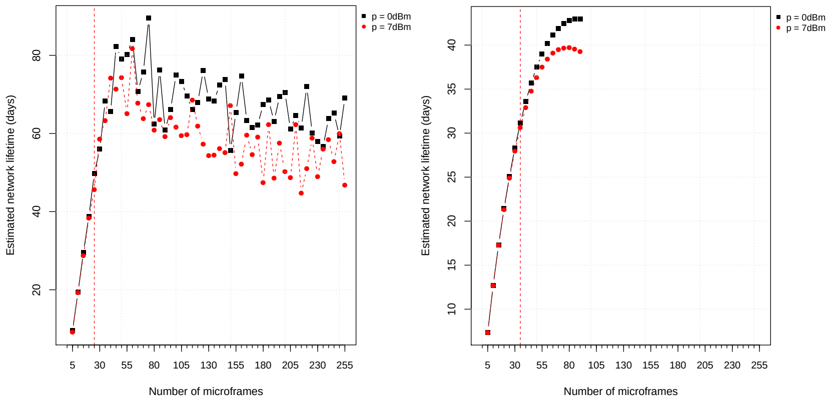
Figure 60: Delivery ratio for selected configurations of LISHA office scenario. In no configuration the MAC is able to deliver 100% of the messages. 21 microframes with p=7dBm is the largest microframe count where it reliably delivers more than 99.99% of messages in time.



(a) Simulation

(b) Analytic

Figure 61: Mean latency for LISHA office scenario. Analytic predictions work better for non-saturated networks. The steep increase in latency is not observed in the simulations, as the analytic model does not consider dropping expired messages.



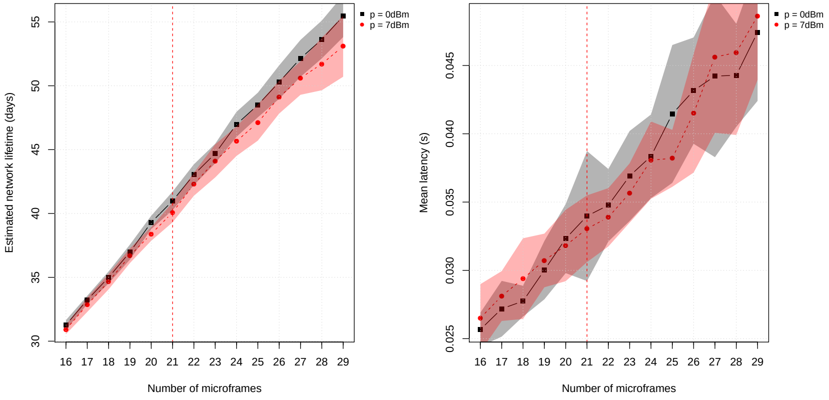
(a) Simulation

(b) Analytic

Figure 62: Estimated network lifetime for LISHA office scenario. The optimum point is around 50 to 80 microframes.

a low delivery ratio (Figure 59), so Figure 63 considers 16 to 30 microframes. Although 7dBm presents a smaller lifetime when compared to the same MAC period at 0dBm, it achieves a slightly better delivery ratio, allowing the increase of the number of microframes slightly beyond what 0dBm supports for the same delivery ratio. The analytic model captures the lifetime behavior, but is pessimistic because it considers that all nodes are battery powered and act as forwarders.

The selected configuration for the LISHA office scenario is summarized in Table 13.



(a) Network lifetime

(b) Mean latency

Figure 63: Estimated network lifetime and mean latency for selected configurations of LISHA office scenario. Before reaching the optimum point, lifetime grows linearly.

Number of nodes	40
Field size	30m x 40m
Data rate (λ)	22.35
Synchronization period	12s

TX power (p)	**0dBm**	**7dBm**
Radio range	18m	35m
Avg. # of hops (H)	1.154	1
Avg. # of neighbors (η)	24.513	37.128
TX power consumption	72mW	102mW

Node configurations					
Nodes with this config.	3	3	10	12	11
Data period (d)	60s	0.3s	1s	10s	10s
Data expiry	0.3s	0.3s	1s	10s	10s
Battery-powered	Yes	No	No	No	Yes
Acts as forwarder	No	Yes	Yes	Yes	No

Table 12: Configurations for SSB office scenario.

4.4.2.3 SSB Office Scenario

The SSB office network has similar parameters to the LISHA variation, but with more nodes deployed over a larger area, and a higher overall data rate. Its configurations are shown in Table 12, and the node map is shown in Figure 64.

Figure 65 shows that TSTP is not able to achieve 100% or even 99.99% of delivery ratio in any configuration. Therefore, the selected configuration is the one that achieves the highest delivery ratio, even at the cost of lifetime (Figure 66). The analytic model did not produce any valid result, as it finds the network saturated with messages in every MAC configuration.

The selected best configurations for both office scenarios are summarized in Table 13. The configurations shown in Tables 10 and 13 are used for the experiments in Sections 4.4.3 and 4.4.4.

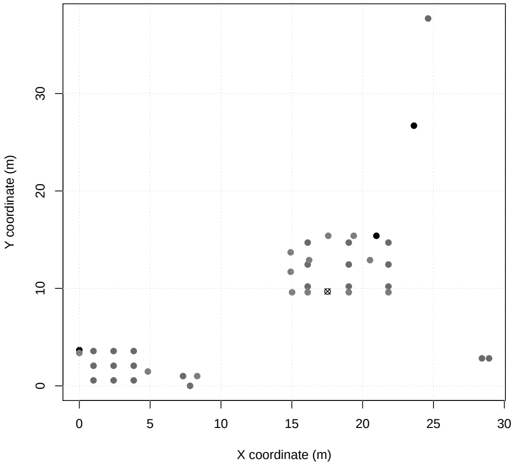
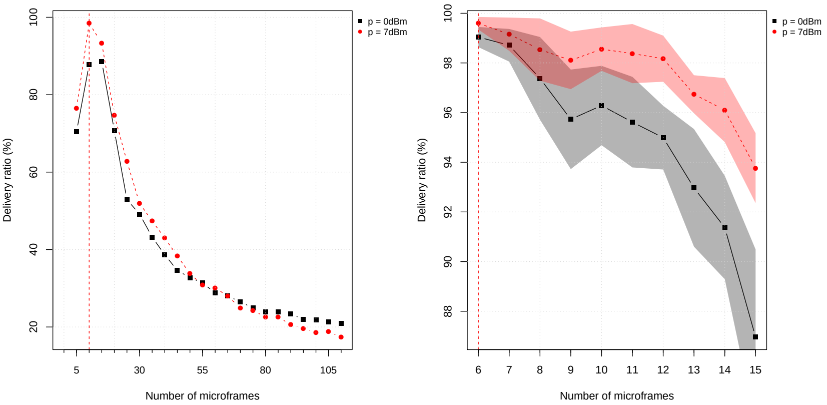


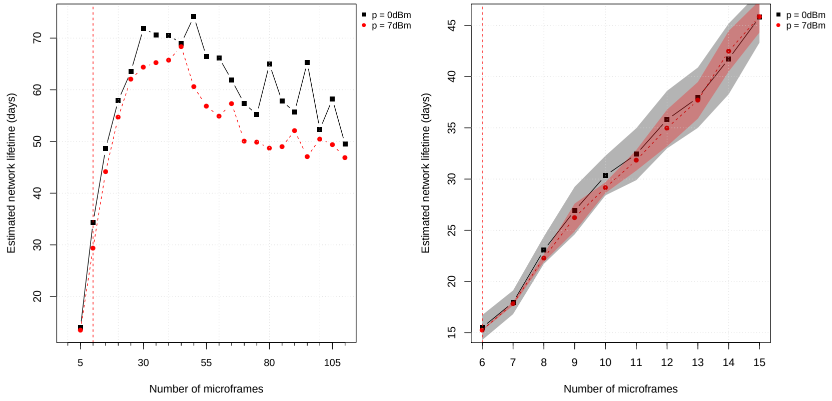
Figure 64: Office SSB node map. Darker shades indicate higher data periods. The sink is marked with an X.



(a) Coarse microframe steps

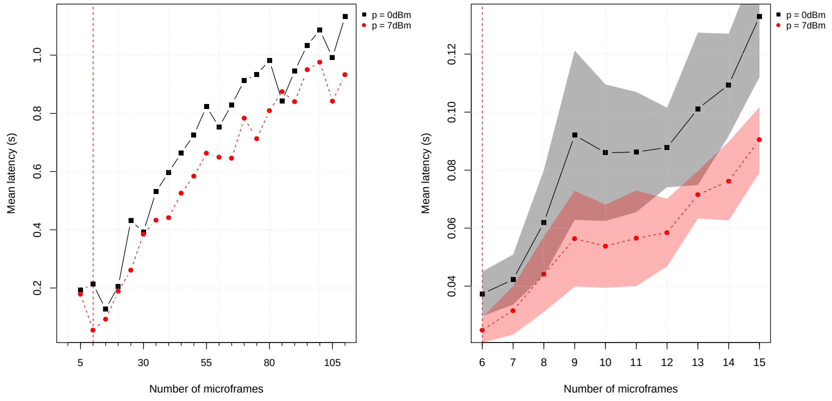
(b) Granular microframe steps

Figure 65: Delivery ratio for SSB office scenario. (a) At 5 microframes, the MAC contention period is too small, leading to too many collisions and a drop in delivery ratio. At 10 microframes, it gets the closest to 100%, and falls from 15 and higher. So the best configuration for delivery ratio must be in the interval [6,14] microframes. (b) The best delivery ratio is at p=7dBm and 6 microframes.



(a) Coarse microframe steps (b) Granular microframe steps

Figure 66: Estimated network lifetime for SSB office scenario. Higher lifetime can be achieved, but only with a significant decrease in delivery ratio.



(a) Coarse microframe steps (b) Granular microframe steps

Figure 67: Mean latency for SSB office scenario. At 5 microframes, the short MAC contention period leads to significantly more collisions and an increase in latency.

Variable	**Unit**		
Node map	-	LISHA	SSB
Nodes	-	14	40
TX power	dBm	7	7
Microframes	-	21	6
MAC period	ms	14.28	3.90
Delivery ratio	%	99.99	99.59
Network life	days	40.08	15.27
Nominal d.c.	%	8.78	32.09
Effective d.c.	%	10.79	27.38
Max latency	ms	495.67	1268.98
Mean latency	ms	33.05	24.85

Table 13: Selected configurations for each office scenario.

4.4.3 Synchronization

The following set of simulations investigates the effectiveness of the employment of speculative synchronization techniques such as the Speculative Precision Time Protocol (Sections 2.4, 3.4) in TSTP. To allow nodes to synchronize only by overhearing passing messages, TSTP includes in the common message header (Figure 17) of *every message* synchronization information such as the precise time of message transmission. This increases the size of every message, but reduces the overall number of messages if compared to an approach that sends synchronization information only in periodic messages commonly known as *beacons*. The speculative approach should save energy in most cases, as sending a new message is much more energy-consuming than including extra bytes in a message that would already be sent, because new messages come with MAC preambles, contention, potential collisions, extra headers, etc. However, depending on the proportion of synchronization beacons and data messages, there may be cases where explicit synchronization approaches perform better. The following set of experiments analyzes if, and under which circunstances, is it better to pigtail timing information to every message, compared to only exchanging timing information in periodic beacons.

The experiments that follow use the same scenario parameters presented in Tables 9, 11, 12, with the identified optimal MAC preamble and radio transmission power configurations shown in Tables 10, 13. The analyzed input variable is the SPTP maximum tolerable synchronization period P (Section 3.4.2), which was fixed in the prior simulations. The impact of this parameter on delivery ratio, latency, network lifetime, and average clock synchronization accuracy is evaluated. Each experiment runs for 2 hours of simulated time, and is replicated 20 times. Shaded areas in the figures represent standard deviations among the means of the 20 replications. For the office scenarios, the considered synchronization periods are 240s, 200s, 160s, 140s, 120s, 100s, 80s, 60s, 40s, 30s, 20s, 10s, 5s. For the environment monitoring scenarios, the considered synchronization periods are 3600s, 2700s, 2400s, 1800s, 1200s, 900s, 600s, 480s, 300s, 240s, 180s, 120s, 60s, 30s, 15s.

TSTP performance (**Speculative**) is compared with a theoretical and an explicit strategies. The **Theoretical** data are calculations considering that timestamps are *not* embedded in every message, but transmitted once per half synchronization period per node. It considers perfect channel conditions with no collisions, representing an approximation of what any explicit synchronization protocol with a similar strategy could achieve[2]. The **Explicit** strategy is a variation on TSTP that implements the theoretical approach: it does not include timestamps in the common message header (making it smaller), but only in `Keep Alive` messages (making them potentially more frequent).

Clock synchronization accuracy is measured once per simulated second by taking the absolute difference between each node's current timestamp and the sink's.

4.4.3.1 LISHA Office Scenario

Figure 68 presents the impact of the synchronization period on delivery ratio. Lower synchronization periods decrease delivery ratio, as more `Keep Alive` messages are sent. The speculative approach shows a generally higher standard deviation, since depending on the environment a node might need to send more or less explicit synchronization messages, whereas for Explicit the number of `Keep Alive`s only depends on the synchronization period. As the period increases, the number of explicit messages decreases in both approaches, and they become more similar. Latency (Figure 69) shows a similar behavior, for the same reasons.

The number of `Keep Alive` messages sent are shown in Figure 70. As expected, the number of explicit synchronization messages for each strategy converges as the synchronization period increases, and the speculative approach reaches zero as the synchronization period gets larger than twice the maximum application data period of 60s. The explicit approach sends more synchronization messages than the theoretical, as the latter considers a perfect network with one synchronization message per node per half period, with no collisions.

Figure 71 shows the average clock error between sensor nodes and the sink. Clock synchronization accuracy is measured once per simulated second by taking the absolute difference between each node's current timestamp and the sink's. In this case, the “Theoretical” line shows the expected clock error to occur in the expected time elapsed between two synchronization messages, assuming that synchronization

[2] It is not meant to be an absolute best case, but a comparable approach with good performance. A protocol could achieve better performance than the Theoretical approach, as it considers that every node sends synchronization beacons at every half period.

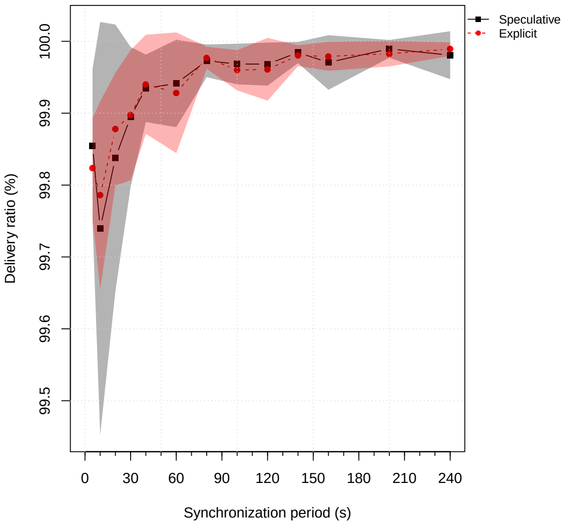


Figure 68: Delivery ratio for LISHA office scenario, p=7dBm, 21 microframes, varying synchronization period. Comparison between TSTP (Speculative), and a TSTP version that does not include timestamps in the header and will only synchronize with explicit keep alives (Explicit). Higher synchronization periods lead to less explicit messages, so delivery ratio grows. As the synchronization period grows, both approaches converge, since there will be fewer keep alives.

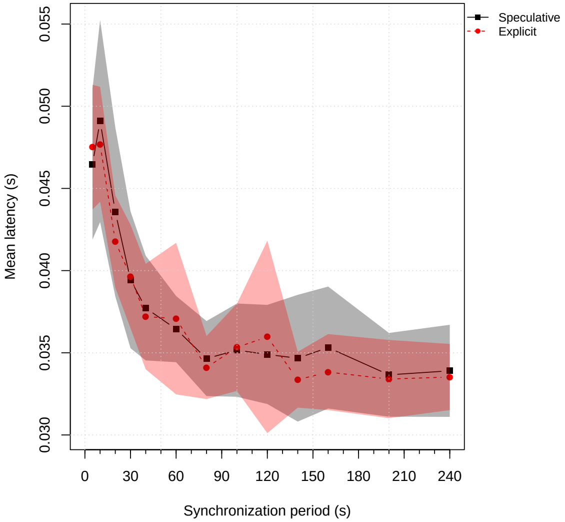


Figure 69: Average latency for LISHA office scenario, p=7dBm, 21 microframes, varying synchronization period. Comparison between TSTP (Speculative), and a TSTP version that does not include timestamps in the header and will only synchronize with explicit keep alives (Explicit). Higher synchronization periods leads to less explicit messages, so mean latency decreases. As the synchronization period grows, both variations converge, since there will be fewer keep alives.

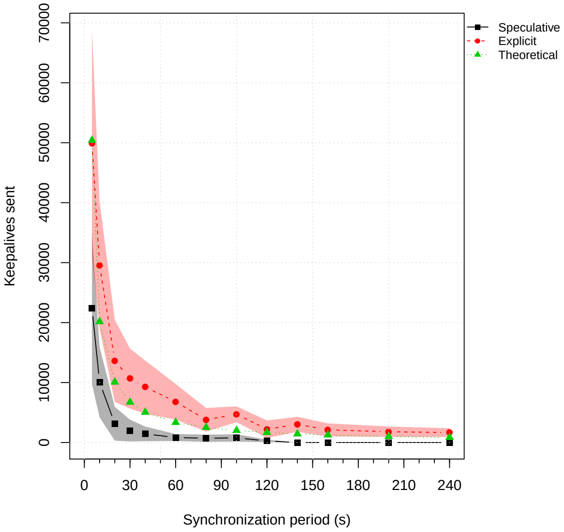


Figure 70: Number of explicit synchronization messages for LISHA office scenario. "Theoretical" is one keep alive per node per synchronization half period. "Explicit" sends more keep alives due to imperfect channel conditions. "Speculative" sends less because it uses regular data messages as synchronization messages most of the time. At periods greater than 140s, Speculative reaches zero explicit synchronization messages.

messages arrive at every node with a period P_S:

$$P_S = min(S_P/2, d) \tag{4.6}$$

where S_P is the synchronization period, and d is the average application data period (12s for the office scenarios). Theoretical clock drifts are generated in the same way as Castalia computes clock frequency errors. This bound applies only to the speculative strategy, as the explicit one triggers synchronization messages only every $S_P/2$ time, and the clock error upper bound grows proportionally to the synchronization period.

The average clock error for both approaches is below the calculated upper bound, indicating that SPTP's frequency error correction (Section 2.4) is effective. As the synchronization period increases, the speculative approach is expected to achieve better clock synchronization than the explicit strategy, because it uses timestamps from regular data messages in addition to the ones in `Keep Alive` messages. The data shows that the difference is not big in this scenario, however. It also shows that the speculative clock error decreases with higher synchronization periods, which can be explained by the higher delivery ratio achieved in such cases: the network is less saturated, so nodes can actually get timestamps without collisions more often.

The analyses so far indicate that there is not a big difference in terms of latency, clock error, or delivery ratio between the explicit and speculative approaches for the LISHA office scenario. As Figure 70 shows, the speculative approach saves a considerable number of `Keep Alive` messages. However, regular data messages are bigger in the speculative approach, as they include timestamps, so an energy analysis is necessary to determine whether the savings in number of messages is bigger than the cost of pigtailing timestamps.

Figure 72 compares the energy consumption of both strategies with the theoretical approach. At low synchronization periods, the speculative approach is able to save a proportionally large number of `Keep Alive` messages, and performs much better in terms of energy. As the proportion of synchronization messages decrease in relation to data messages, the cost of pigtailing timestamps to data messages is proportionally bigger, and the speculative approach performs comparatively worse. At a synchronization period of around 90s, the theoretical approach starts saving more energy than the speculative one. However, the explicit strategy, which is an *implementation* of the theoretical approach subjected to similar network conditions, shows that the speculative strategy saves more energy for all synchronization periods up

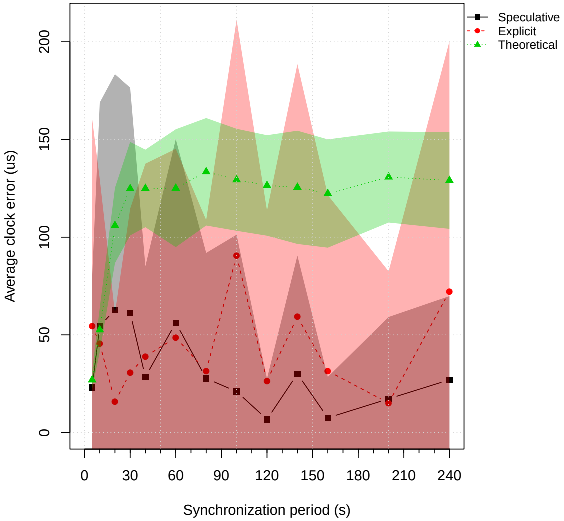


Figure 71: Average clock error for LISHA office scenario. Clock error should be inversely proportional to synchronization period. However, as the synchronization period grows, delivery ratio also grows, and this indicates that nodes are able to access more synchronization messages more often, so clock error is actually lower at higher synchronization periods. For periods >160s (for Speculative) and >200s (for Explicit), delivery ratio stabilizes and clock error grows according to the sync period.

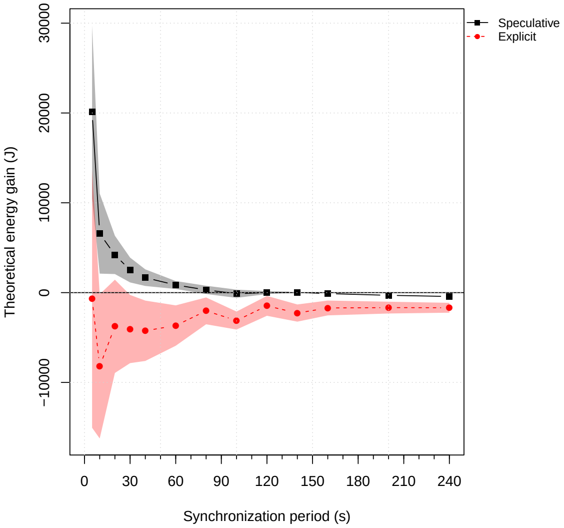


Figure 72: Energy consumption compared to theoretical explicit approach for LISHA office scenario. Theoretical energy gain is calculated comparing the cost of including timestamps in every message to the savings in number of explicit synchronization messages.

to 240s (four times the maximum application data period, 60s). Figure 73 shows the estimated network lifetime for both implementations, confirming that the speculative approach saves significant amounts of energy in every case.

4.4.3.2 SSB Office Scenario

The SSB office scenario features more nodes and hops than the LISHA scenario, with a high data rate that puts the network over its limit. Figures 74 and 75 show that the speculative approach performs slightly better than the explicit approach in terms of delivery ratio and latency. Delivery ratio increases and average latency decreases for both implementations as the synchronization period gets bigger.

The number of Keep Alives sent, shown in Figure 76, shows the same behavior as in the LISHA scenario. The average clock error (Figure 77) is slightly better for the speculative implementation, as the denser network deployment allows nodes to receive data messages more

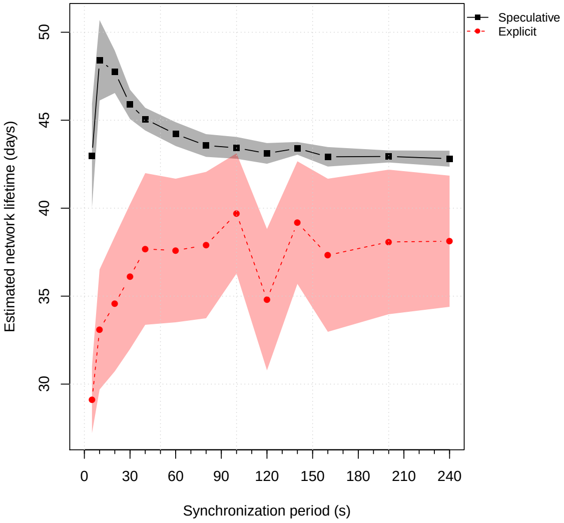


Figure 73: Estimated network lifetime for LISHA office scenario. The Speculative approach saves energy significatively.

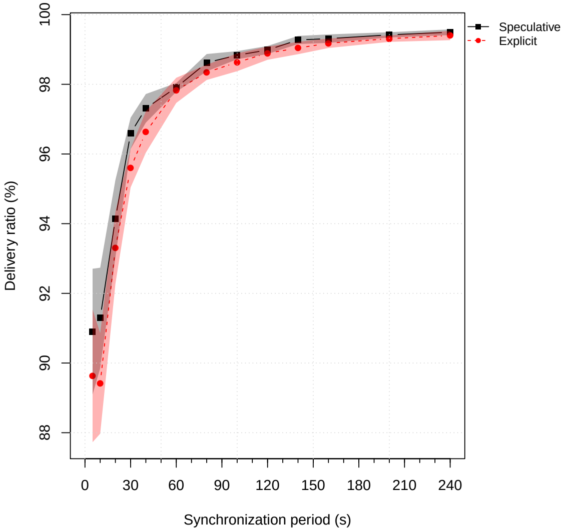


Figure 74: Delivery ratio for SSB office scenario, with p=7dBm and 6 microframes. Similarly to the LISHA variation, delivery ratio grows as less synchronization messages are generated.

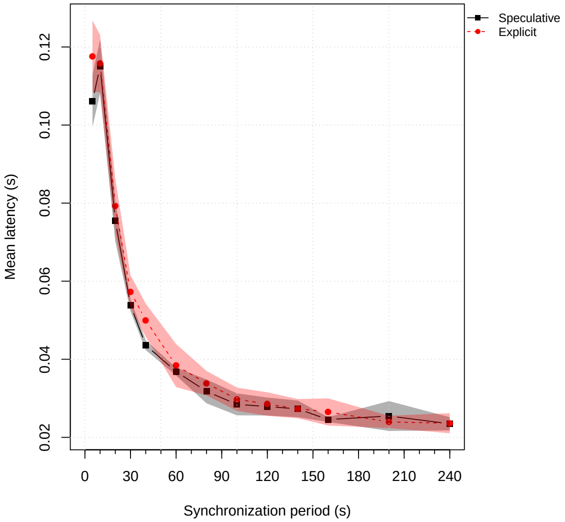


Figure 75: Average latency for SSB office scenario. Similarly to the LISHA variation, latency decreases as less synchronization messages are generated.

frequently.

Figure 78 shows that a theoretical implementation could reach lower overall energy consumption starting from a synchronization period of around 10 seconds. However, in practice, the explicit implementation still spends more energy on explicit synchronization messages than it saves in not including timestamps in the common header. Figure 79 shows that this energy difference leads to a very small variation in the estimated network lifetime, possibly because the battery-powered nodes are not particularly benefited from the energy savings.

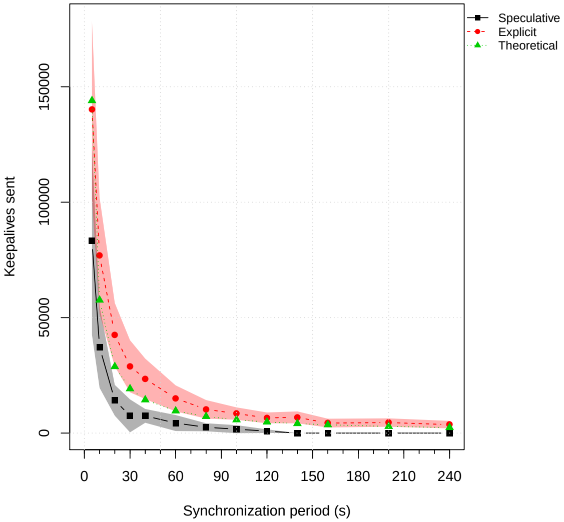


Figure 76: Number of explicit synchronization messages for SSB office scenario. “Theoretical” is one keep alive per node per half synchronization period. “Explicit” sends more keep alives due to imperfect channel conditions. “Speculative” sends less because it uses regular data messages as synchronization messages most of the time. At periods greater than 140s, Speculative reaches zero explicit synchronization messages.

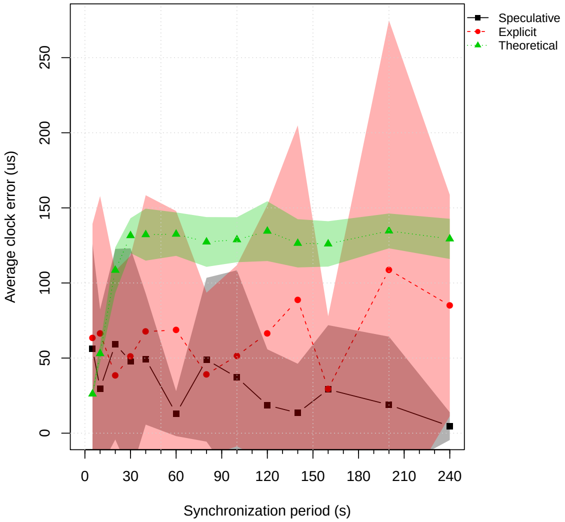


Figure 77: Average clock error for SSB office scenario. Clock error should be inversely proportional to synchronization period. However, as the synchronization period grows, delivery ratio also grows, and this indicates that nodes are able to access more synchronization messages more often, so clock error is actually lower at higher synchronization periods.

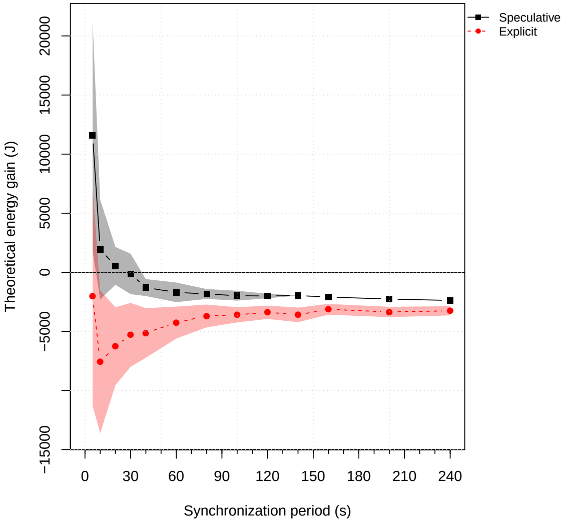


Figure 78: Energy consumption compared to theoretical explicit approach for SSB office scenario. Theoretical energy gain is calculated comparing the cost of including timestamps in every message to the savings in number of explicit synchronization messages. The theoretical approach quickly outperforms the speculative, which is still better than "Explicit", an implementation of the theoretical approach measured under the same conditions as "Speculative".

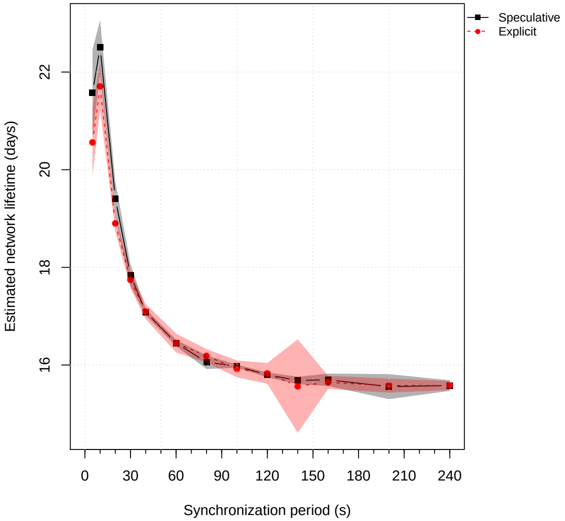


Figure 79: Estimated network lifetime for SSB office scenario. Both approaches show similar results.

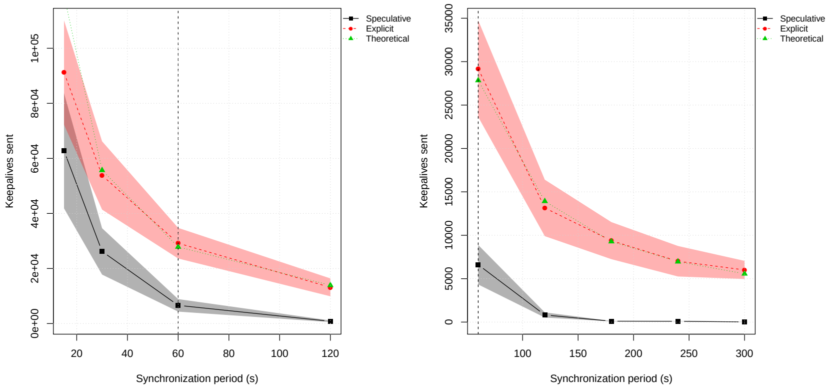
In the LISHA office scenario, the speculative and explicit approaches perform very close in terms of overall clock error, latency, and delivery ratio, but the speculative approach shows significantly lower energy consumption. In the SSB scenario where the network is saturated, the speculative approach performs only slightly better, but does so for every measured metric.

4.4.3.3 Environment Monitoring Scenario

For better visualization, the graphics that follow are divided in two: low synchronization periods, up to $2d$ (where d is the application data period), and high synchronization periods, from d up to $6d$. The dashed vertical lines mark the point in the x axis where the synchronization period is equal to the data period. Data periods of 60s and 600s are considered in this section. The results for the periods of 300s and 900s show similar behavior to the 600s ones, and can be found in Section A.1 in the Appendix.

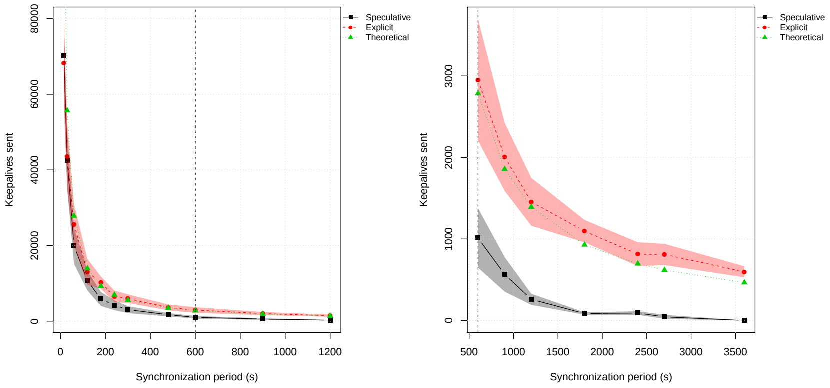
The environment monitoring scenario is a more regular one than both office scenarios. The network is larger, node deployment is uniform, and traffic is sparser, which makes the predictions of the theoretical approach more accurate. Figures 80, 81 show that indeed the number of synchronization messages sent by the explicit approach is close to one per node per half period (the “Theoretical” line). The speculative approach generally saves a large number of explicit messages, unless the synchronization period is much lower than the data period, in which case there are not enough data messages to serve as synchronization messages. Figure 81a shows that this only starts happening when the synchronization period falls below around $d/20$.

Latency is expected to start lower with the speculative approach, as it is able to send less messages than the explicit one. However, the increase in data message size caused by the inclusion of timestamps eventually leads to a slightly higher latency, as the synchronization period gets higher and the network produces much more data messages than synchronization messages. Figures 82, 83 show that the synchronization period needs to be around 5 times the data period for the explicit approach to start showing slightly better latencies.



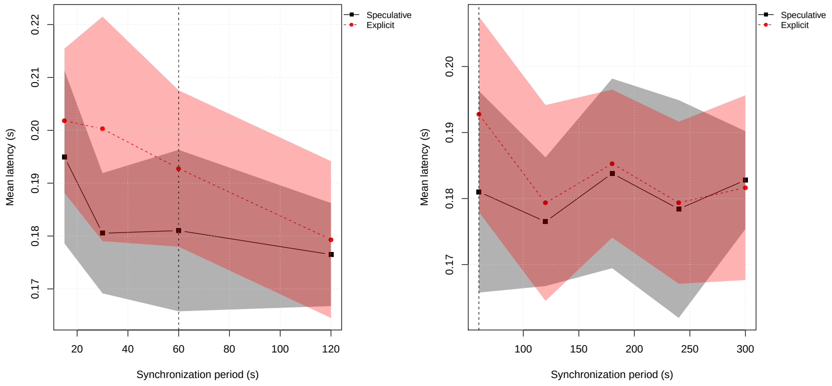
(a) Low synchronization periods (b) High synchronization periods

Figure 80: Number of explicit synchronization messages for environment monitoring scenario, p=7dBm, d=60s. “Theoretical” is one keep alive per node per half synchronization period. “Explicit” sends more keep alives due to imperfect channel conditions. “Speculative” sends less because it uses regular data messages as synchronization messages most of the time.



(a) Low synchronization periods (b) High synchronization periods

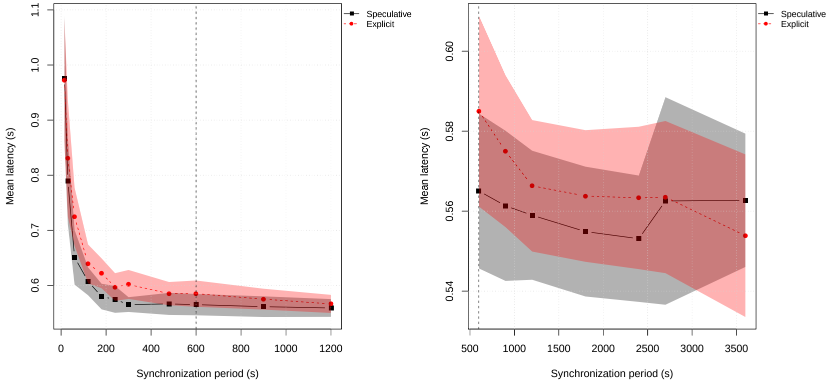
Figure 81: Number of explicit synchronization messages for environment monitoring scenario, p=7dBm, d=600s.



(a) Low synchronization periods

(b) High synchronization periods

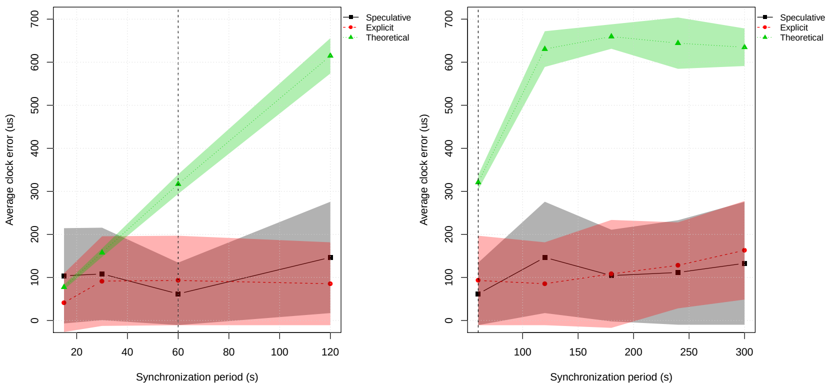
Figure 82: Average latency for environment monitoring scenario, d=60s. Only at synchronization periods of 300s the Explicit approach reaches a slightly better average.



(a) Low synchronization periods

(b) High synchronization periods

Figure 83: Average latency for environment monitoring scenario, d=600s. Only at synchronization periods of 3600s the Explicit approach reaches a slightly better average.

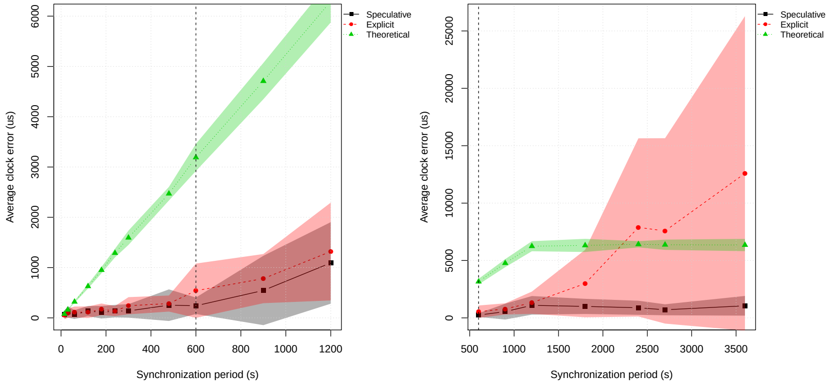


(a) Low synchronization periods (b) High synchronization periods

Figure 84: Average clock error for environment monitoring scenario, d=60s.

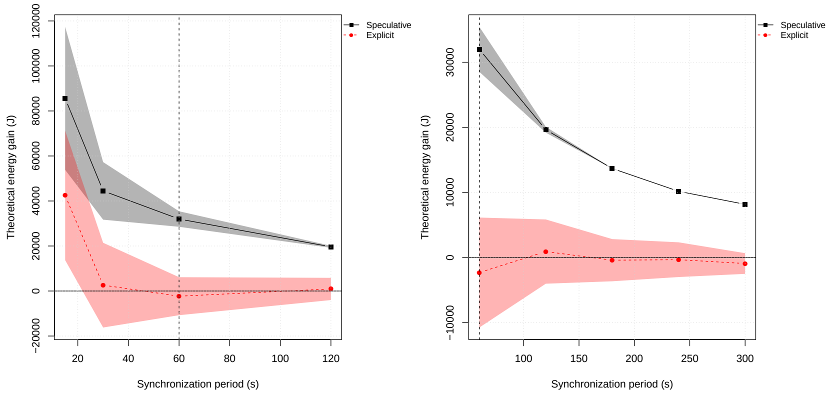
Increasing the synchronization period, however, necessarily increases the overall clock error for the explicit strategy. Figures 84 and 85 show that both strategies are roughly equally effective at synchronizing the clocks. However, with synchronization periods greater than 1500s, the explicit approach starts to wildly vary its clock synchronization effectiveness, while the speculative approach maintains constant performance as the data period remains the same.

Figures 86 and 87 further confirm that the theoretical estimations are more accurate for the environment monitoring scenarios, as the explicit implementation shows an energy performance close to the theoretical. The Figures also show that the overall energy consumption for the speculative approach is lower in every measurement. Figures 88, 89 show that the expected network lifetime for the speculative approach is indeed higher up until synchronization periods of around $5d$.



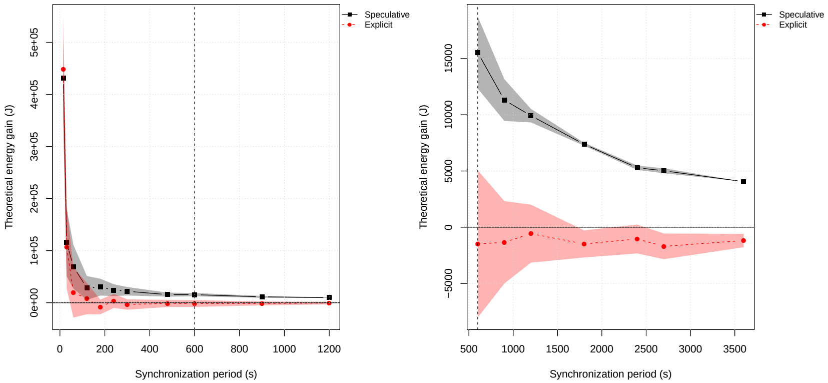
(a) Low synchronization periods (b) High synchronization periods

Figure 85: Average clock error for environment monitoring scenario, d=600s.



(a) Low synchronization periods (b) High synchronization periods

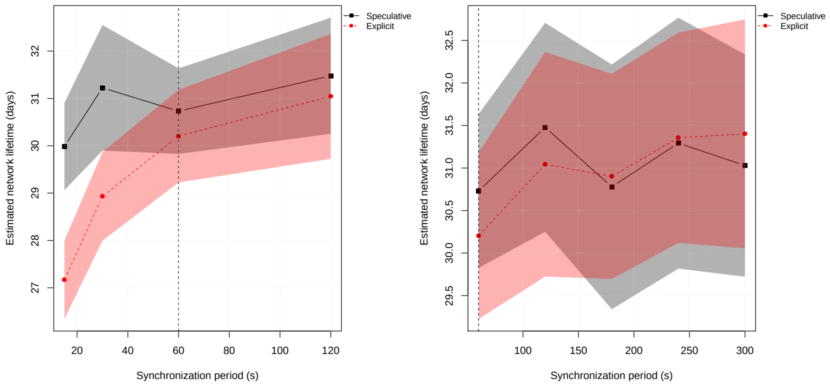
Figure 86: Energy consumption compared to theoretical explicit approach for environment monitoring scenario, d=60s.



(a) Low synchronization periods

(b) High synchronization periods

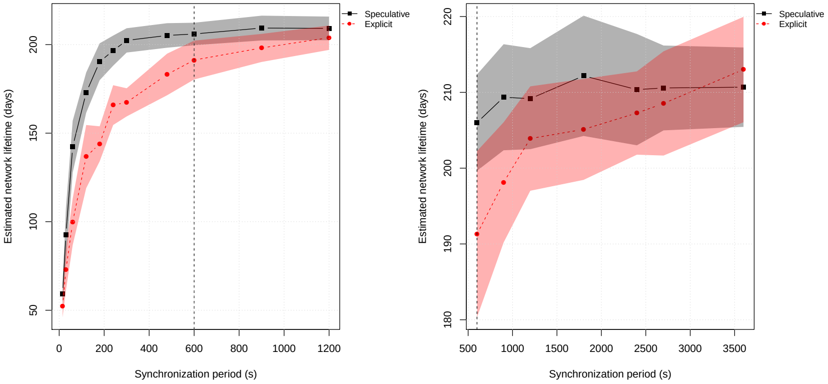
Figure 87: Energy consumption compared to theoretical explicit approach for environment monitoring scenario, d=600s.



(a) Low synchronization periods (b) High synchronization periods

Figure 88: Estimated network lifetime for environment monitoring scenario, d=60s. Only at synchronization periods of 300s the Explicit approach reaches a slightly better average.

The analyses show that, for the environment monitoring scenarios, the speculative approach shows better latency and energy consumption, and comparable clock error in relation to the explicit synchronization approach, which performs close to a theoretical explicit clock synchronization algorithm. When the synchronization period is greater than five times the data period, the burden of including timestamps in each message starts to show on energy and latency. However, the speculative approach maintains clock synchronization performance independent of the synchronization period, and for higher data periods the $5d$ point where the explicit approach becomes better comes with the cost of a *large* increase in clock error across the network.



(a) Low synchronization periods

(b) High synchronization periods

Figure 89: Estimated network lifetime for environment monitoring scenario, d=600s. Only at synchronization periods of 3600s the Explicit approach reaches a slightly better average.

4.4.4 Routing Metrics

Equation 2.7 in Section 2.5.1 introduces the concept of the distortion coefficient α, which is a real number in the interval $[0, 1]$ used to skew the distance-based contention offset for message transmission according to any metric of interest. In this section, two such metrics are presented and their overall impact on network performance is evaluated.

4.4.4.1 Expiry Metric

The Expiry metric defines a distortion coefficient influenced by how much time the message has left to reach its destination. Messages that are close to expiring shrink space, reducing the node's offset $\delta(m)$, and providing it a better chance to win the contention and forward that message quickly. Equation 4.7 defines the distortion coefficient for the Expiry metric.

$$\alpha = \frac{E_m - t}{E_m} \tag{4.7}$$

where E_m is the timestamp in which message m expires and t is the current time. It is assumed that $E_m > t$, as otherwise m is an expired message that should be dropped. The Expiry metric has been used for all of the simulation experiments presented so far.

4.4.4.2 Effort Metric

The Effort metric distorts space based on how much the node has cooperated with the network doing message forwarding in the past. The more messages the node has forwarded, the more space is dilated, and the less likely that node becomes to forward again. This metric seeks a balance on the network for neighbors on the same hop from the destination to better alternate the effort of forwarding messages. Equation 4.8 defines the distortion coefficient for the effort metric.

$$\alpha = \frac{F_n}{Tx_n} \tag{4.8}$$

where F_n is the number of queued messages from neighbors that node n has relayed so far, and Tx_n is the total number of messages that the node has transmitted in the past (including its own generated messages). The effort metric takes information from the node's history, rather than from the message being forwarded.

4.4.4.3 Evaluation

The goal of the following experiments is to analyze the effectiveness of each different routing metric. The metrics are compared to three other control groups. The **Distance** metric takes only the distance into account, and is defined as Equation 2.7 with a fixed $\alpha = 1$. Likewise, the **Zero** metric simply sets $\alpha = 0$, such that nodes always have the lowest possible offset, and collision control is fully managed by the MAC's random backoff and silence mechanisms (Section 3.5.1). **Random** sets every offset δ to a random number. A combination of the Expiry and Effort metrics is also assessed (the two α values multiplied), and is labeled **Expiry Effort Distance**. The **Expiry** and **Effort** metrics are explained in Sections 4.4.4.1 and 4.4.4.2, respectively.

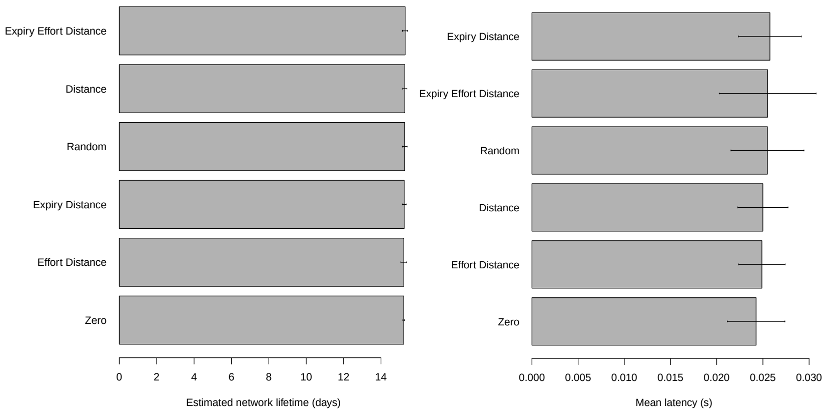
The same scenario parameters presented in Tables 9, 11, 12 are used, with the identified optimal MAC preamble and radio transmission power configurations shown in Tables 10, 13. The varied input variable is the way the α component is calculated during MAC contention.

The statistics measured are network lifetime, average latency, transmission fairness index, and offset standard deviation. The fairness index indicates how balanced was the amount of messages that were relayed by the nodes, according to Equation 4.9:

$$FI = \frac{(\sum F)^2}{N \times \sum(F^2)} \tag{4.9}$$

where F is the number of message relayed, and N is the number of nodes. The highest the value of the fairness index, the more nodes alternate in forwarding messages, which leads to a better distribution of the energy consumption, avoiding a certain routing path to be overused and prematurely deplete the energy of particular nodes. Because by design the sink will normally spend much more energy than any other node (it needs to acknowledge every message), its message transmission statistics are not included in the fairness index calculation. The offset standard deviation indicates how well the metric spreads offsets, potentially reducing the probability of collisions.

For the SSB office scenario, Figure 90a shows that the rout-



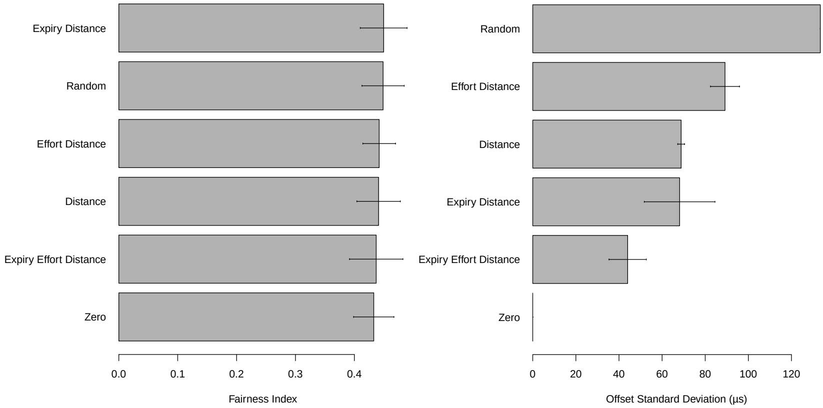
(a) Network lifetime

(b) Mean latency

Figure 90: Estimated network lifetime and mean latency for SSB office scenario under different routing metrics.

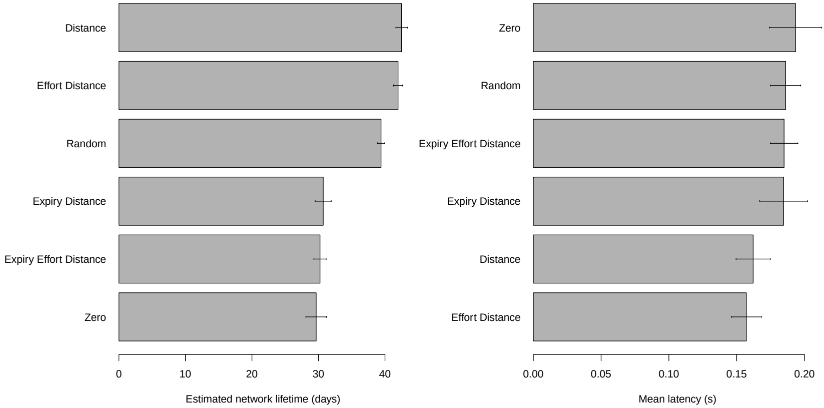
ing metrics have little impact on network life. This is expected, as battery-powered nodes do not forward messages in the office scenarios. Figures 90b and 91a suggest that the variations in latency and fairness index are not statistically relevant, as there is a big overlap between all standard deviation intervals. This happens because SSB is a scenario with few hops overall, such that routing decisions have little impact.

In the larger environment monitoring scenario, messages need to travel more hops on average, and routing decisions impact network performance. Figure 92a shows that the Distance and Effort metrics yield the best network lifetime for a data period of 60s. The Distance metric calculates larger offsets in general, which means that nodes spend more time with their radios off before transmission. However, for larger data periods, the configured MAC contention period that limits the offsets is increased (Table 10), and the routing metrics have more room to vary the offset without causing collisions. Figure 94a shows that for $d = 600s$, the Effort metric performs the best in terms of network lifetime. It is the only metric that significantly outperforms Random, which also achieves a balancing in forwarder selection. As expected, the Zero metric shows a comparatively bad network lifetime, as it is more likely to cause collisions and retransmissions. The fact that both Expiry metrics are close to the Zero metric for d=60s and close to the Random metric in d=600s further indicates that the MAC period of 29.51ms is too short for these metrics, generating more collisions.



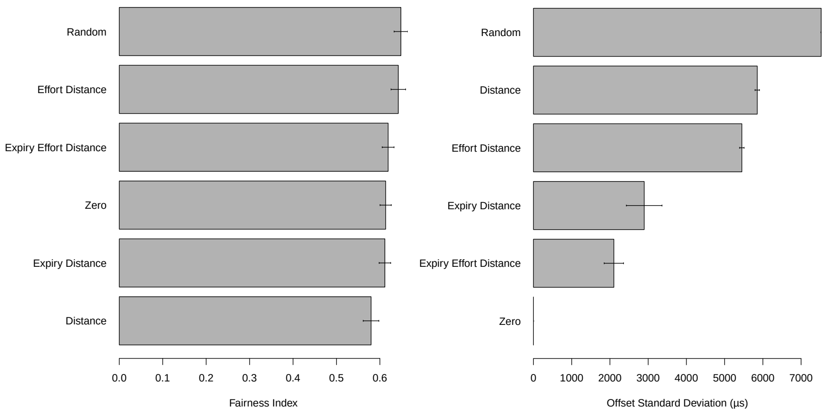
(a) Fainess index (b) Offset standard deviation

Figure 91: Fairness index and offset standard deviation for SSB office scenario under different routing metrics.



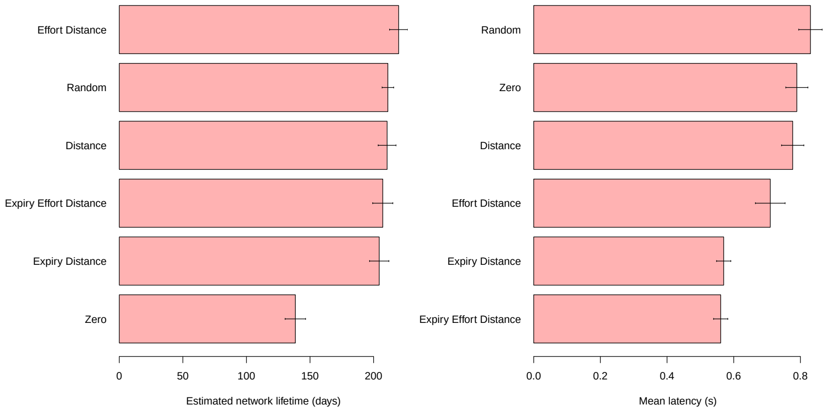
(a) Network lifetime (b) Mean latency

Figure 92: Estimated network lifetime and mean latency for environment monitoring scenario, d=60s, under different routing metrics.



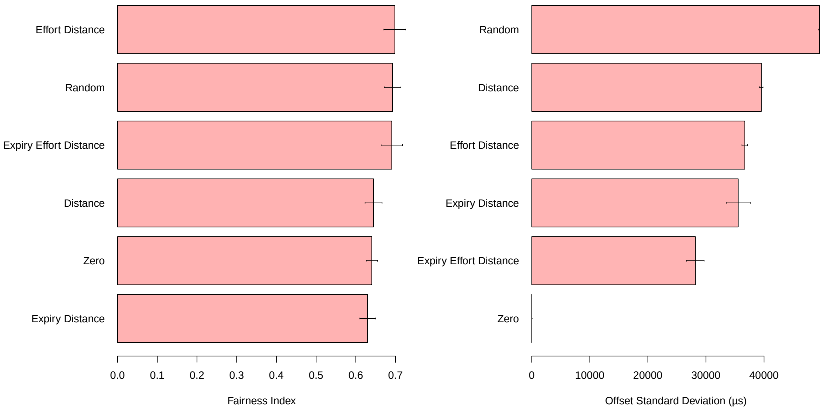
(a) Fainess index (b) Offset standard deviation

Figure 93: Fairness index and offset standard deviation for environment monitoring scenario, d=60s, under different routing metrics.



(a) Network lifetime (b) Mean latency

Figure 94: Estimated network lifetime and mean latency for environment monitoring scenario, d=600s, under different routing metrics.



(a) Fainess index

(b) Offset standard deviation

Figure 95: Fairness index and offset standard deviation for environment monitoring scenario, d=600s, under different routing metrics.

Figures 93a, 95a indicate that the Effort metric is indeed able to maintain a high fairness index, comparable to the Random metric. Figure 92b shows that the Distance and Effort metrics end up having a lower latency when d=60s, possibly for avoiding collisions more than the other metrics. However, for a higher data and MAC period, the Expiry metric significantly reduces the overall latency, as shown in Figure 94b.

No strong correlations were found between the offset standard deviations (Figures 91b, 93b, 95b) and other statistics, which indicates that a higher spreading of the offset does not significantly reduce the number of collisions in the observed cases.

Results for the LISHA office scenario and environment monitoring with d=300s and d=900s are similar to SSB and d=600s, and can be found in Section A.2 of the Appendix.

Overall, the Expiry metric performs better in terms of latency and only slightly worse in terms of network lifetime than the Effort metric. However, for smaller MAC periods, the Effort metric is able to prevent more collisions and balance the traffic better, performing better even in terms of latency.

4.5 EVALUATION SUMMARY

The main results obtained from the performance evaluations are:

- The code sizes for both implementations of TSTP (simulator and real-world), including all of its features, is close to two third-party implementations of the AODV protocol for the same simulation framework. AODV only implements routing.

- The buffer handling architecture shows light time overhead. The average time overhead for allocation of zero-copy buffers in EPOSMote III is 2.31μs. The notification mechanism used for buffer propagation accounts for 11.60% of the total buffer processing time across the components.

- The code responsible for time synchronization on EPOSMote III is time-deterministic. The delay from sending and receiving Start-of-Frame Delimiters (SFD) between two devices has a jitter of 93.5ns, including software and radio delays. This leads to a highly accurate time synchronization protocol that is able to achieve sub-microsecond instant clock synchronization, as measured on real EPOSMote III devices.

- There is an optimal point on MAC preamble size which depends on network topology and traffic. By selecting the right MAC parameters, large networks are able to achieve effective radio duty cycles smaller than 1% while keeping 100% delivery ratio and average latencies around 0.5s.

- Including timestamps in every message to enable speculative clock synchronization is beneficial in all the scenarios considered when compared to explicit, beaconed clock synchronization strategies. For small, traffic-intensive networks, the reduction in number of messages leads to an increase in energy efficiency. For larger networks with lighter traffic, the inclusion of timestamps in data messages slightly improves latencies and energy efficiency, while keeping a better lower bound on clock accuracy.

- When the network is not saturated, the simplistic analytic model shows similar results to the detailed simulations, while being much faster to execute. It is thus an appropriate first step for analyzing the expected behavior of new network deployments.

4.6 DISCUSSION

The MAC currently in use by TSTP, based on RB-MAC, has many benefits such as low idle listening duty cycles; resiliency to lossy links due to multiple nodes being able to forward a given message, further improved by the possibility of using multiple channels with no additional overhead (Section 3.5.1); enabling reactive routing and speculative synchronization; and allowing energy-constrained nodes to save energy by not acting as forwarders. It also does not impose any requirements on clock synchronization accuracy, nor requires scheduling or structuring of the network. However, the analyses show that its performance in terms of delivery ratio, energy, and latency, makes it unsuitable depending on the network requirements.

For the environment monitoring scenarios, while a delivery ratio of 100% is achievable for all considered data periods, network lifetime is generally the most important metric of interest, and as was shown in the figures, there is an optimal point imposed by the MAC that is dependent on the network and the traffic. This optimum might reach 264 days of full network operation (Table 10) for very sparse traffic, which might be reasonable for some applications, but a lifetime of around one month is not impressive for a network with nodes each generating data once per minute. What's worse, the analyses show that reducing the radio transmission power actually *decreases* network lifetime even more, as messages need to travel more hops towards the destination. The literature commonly refers to 1% as the goal for radio duty cycles, and TSTP MAC is only able to achieve 1% at data periods greater than 300s for the studied scenarios. Nevertheless, if the granted network lifetime is enough for a given application, TSTP achieves reasonably low and bounded latencies, 100% delivery ratio, and clock synchronizations of around hundreds of microseconds across the entire network.

For both office scenarios, the situation is worse. The presence of battery-powered nodes, a noisy environment, higher data rates, and message expiration times as low as 300ms make it impossible for the MAC to reliably deliver 100% of the messages in time for the considered deployments. It seems that this limitation can only be solved by relaxing application requirements: either allowing a margin of message loss, or reducing the overall data rate. While a tolerable message loss does lead to significant gains in network lifetime, such a trade-off between network life and delivery ratio might not be a reasonable imposition on the network design.

Many techniques were employed to improve the MAC's performance (Section 3.5.1), such as switching channels to transmit data messages, making battery-powered nodes not act as forwarders, different buffer selection criteria on the MAC transmission schedule, or manipulating the contention offsets using different metrics. While some of these techniques did in fact improve performance, their result is what is shown in the analyses of this work. It seems like RB-MAC has been tuned to its limit, and it is still not enough for some real-world applications.

The proposed analytic model (Section 4.3) reproduced the general behavior of the MAC, but is not RB-MAC specific. It considers a simple model of collision resolution and makes few assumptions about MAC operation. Namely, it assumes a CSMA/CA model where contention access periods of a given duration arrive on a fixed interval at each node, that only one message can be sent by period (because of the long preambles), and takes as input the average end-to-end throughput of the MAC. Although the model is optimistic on most of its predictions in relation to the observed simulation results, it is not far off, indicating that the specific design and implementation of the MAC are not the major factors in performance, but rather the general techniques employed.

It is an important factor that TSTP as a whole achieves performance close to the analytic model of the MAC while delivering all of its higher-level features that directly benefit applications and programmers (Section 2.1). It was shown that the implementation techniques used for TSTP lead to a code size comparable to an implementation of AODV, a routing-only protocol (Section 4.2.1). The mechanisms for buffer management impose a low overhead in the architecture (Section 4.2.2). The clock synchronization algorithm was shown to be effective, both on real hardware (Section 4.2.3) and across a large simulated network (Section 4.4.3). The contention offset manipulation formula successfully improves different aspects of network performance in certain cases (Section 4.4.4). Moreover, it was shown that the technique of pigtailing timestamps to every message, allowing for speculative time synchronization, is in most scenarios a better technique than sending regular synchronization beacons (Section 4.4.3). Solving the performance limitations of the lower MAC component would not affect these positive results in principle.

4.6.1 Possibilities for MAC improvements

To make TSTP adequate to applications with stricter performance requirements, a possible solution is a major re-design of the MAC. This section looks at related work in MAC protocols to identify possible strategies to improve TSTP MAC's performance. Every design is measured against the many advantages that RB-MAC provides, some of which are pre-requisites for TSTP's speculative synchronization strategies.

RB-MAC defines the cooperative, geographic forwarder selection and passive acknowledgment mechanisms used in TSTP MAC. The authors of RB-MAC have analyzed the protocol in terms of energy consumption and latency, comparing it to 1-hopMAC, where senders define a specific receiver at each hop. They show that RB-MAC is in general more energy efficient and causes less latency because its anycast nature makes it much more resilient to lossy links (AKHAVAN; WATTEYNE; AGHVAMI, 2011). If a link fails, other neighbors can carry on the transmission, incurring no data loss or retransmission costs. Furthermore, not defining a receiver at the MAC level allows for dynamic routing metrics according to particular needs of different nodes.

Dividing the preamble in microframes is also a good characteristic of RB-MAC, which in principle makes it much more energy-efficient than continuous preamble protocols. In another work (STEINER et al., 2013), RB-MAC has been compared to B-MAC (a continuous preamble MAC protocol) and shown to far outperform it for varying channel conditions.

In ContikiMAC (DUNKELS, 2011), senders transmit the data packet repeatedly for a full period instead of using microframes. If the packet is a unicast, the receiver cuts the transmission by sending back an acknowledgment as soon as it wakes up and receives the packet. If this happens, the sender and receiver lock their sleeping phases, resulting in much less repetitions of the data packet in the next unicast transmission involving the same two nodes. Although this results in savings in latency and transmission cost in specific cases, it only works for unicast.

A technique used in many related works to improve performance is making receivers able to signal that they are ready to receive data, so senders can cut the rest of the preamble for the current transmission and transmit the data immediately. However, this technique reduces the RB-MAC forwarder redundancy and makes it harder to employ speculative synchronization techniques, as messages can often be transmitted through the neighborhood of a node without it ever getting a chance to receive it.

CMAC (LIU; FAN; SINHA, 2007) employs the preamble-cutting technique while using the same routing-metric idea as TSTP. Messages can be sent in unicast or anycast modes. In anycast, when a receiver closer to the destination detects a microframe, it backs off for a period proportional to their distance to the destination. Then, if the channel is free, it sends back a CTS, and the sender immediately transmits the data packet. If the receiver that replied with a CTS is in a good enough region, the sender locks in and will start unicasting to this node in future transmissions until one of them fails; if this happens, the sender returns to anycast mode. CMAC equally loses some resiliency to lossy links, and makes it very hard for nodes to passively synchronize to others that are locked in unicast mode.

Given that TSTP is able to keep a whole network synchcronized with measurable accuracy, a promising technique is leveraging this clock synchronization in the MAC itself. Also, leveraging knowledge about the network deployment may lead to performance improvements, although a convenient feature of RB-MAC is operating in a completely ad-hoc manner, and not requiring any clock synchronization.

WiseMAC (EL-HOIYDI; DECOTIGNIE, 2004) is an early MAC protocol that uses the idea of clock synchronization to send packets only when the receiver is likely to be awake. WiseMAC imposes a strong requirement on network topology, assuming that a central node C without energy constraints can reach every node under its supervision and learn those nodes' sleeping schedules by listening to the channel all the time. Then, when the traffic goes from C to one of its children c, C can start sending the preamble only when c will be awake (compensating for the maximum possible clock drift), thus occupying the channel for much less time, providing better throughput and energy savings.

Glossy (FERRARI et al., 2011) is an efficient one-to-all communication architecture for Wireless Sensor Networks that leverages clock synchronization and network structuring with a different approach. It is a cross-layer solution comprising routing, MAC, time synchronization, and network API. Similar to TSTP's Speculative Precision Time Protocol, Glossy exploits control over the MAC layer to synchronize clocks over the network with low cost and high accuracy as a consequence of well-defined communication timings. Glossy exploits constructive interference[3] on IEEE 802.15.4 networks to flood packets with high en-

[3] The phenomenon that two packet transmissions with the same content and very close in time will amplify one another instead of destroying.

ergy and time efficiency, but achieving constructive interference brings very strict timing requirements. The authors detail the techniques used to reach a highly time-deterministic implementation, however the timing requirements bring an inherent probability of transmission failure, as nodes slightly skewed in time will make interference destructive. In Glossy, every transmission is a network-wide broadcast, enabling speculative synchronization, providing high resiliency to lossy links, and implicitly solving routing. However, as no two broadcasts can occupy the channel at the same time, Glossy requires a global scheduling of transmissions in the network.

LWB (FERRARI et al., 2012) extends Glossy to the context of data-gathering WSNs, with many nodes initiating data transmissions. Glossy and LWB present many, but not all of TSTP's features: high accuracy time synchronization; reactive, fully distributed, topology-agnostic routing; and a low power MAC with high redundancy. However, multiple different Glossy floods happening at once would greatly interfere with each other, and so LWB uses a global schedule to ensure that no two floods overlap in time. This global scheduling requires knowledge about all the devices on a network, weakening the advantages of ad-hoc routing.

The DMAC (LU; KRISHNAMACHARI; RAGHAVENDRA, 2004) protocol uses a *staggered schedule* concept, which leverages a known network structure to greatly reduce latency in the uplink direction (traffic flowing from sensors to the sink). It offsets the periodic channel checks such that nodes n-hops away from the sink will wake up right after nodes $(n+1)$-hops away, so nodes in the farther region will start transmission and have a receiver available right away. This design drastically reduces latency all the way from sensors to the sink. Since TSTP provides nodes with their spatial locations, a similar idea might be employable.

As future work for MAC improvement, it seems promising to investigate a time-synchronous MAC design enabled by TSTP's accurate clock synchronization, as well as leveraging spatial knowledge about network deployment given by TSTP coordinates. Other choices of physical layer would also have a significant impact on the performance bottlenecks, as a major part of TSTP's MAC parameters are specifically derived from characteristics of an IEEE 802.15.4 physical layer.

5 CONCLUSION

The Trustful Space-Time Protocol project as an academic endeavour pushes the development of non-traditional solutions to Wireless Sensor Networks. It proposes a novel way of interacting with sensor and actuator devices in terms that make sense in the real world. The principles of *time*, *space*, and *trust* infiltrate the way *data* is handled at all levels, from measurement, to transport, to storage, to visualization.

This work has provided many contributions to the Trustful Space-Time Protocol project. Although every sub-protocol that is part of TSTP has existed in some form in the past, this is the first time the protocol as a whole has been documented in detail, implemented, and evaluated. Many different, reusable tools were developed in conjunction with the protocol's implementation to aid development, debugging, validation, and design of new network deployments.

TSTP today powers two smart rooms and a network of hydrologic monitoring stations that feed a sophisticated IoT architecture with verifiably authentic data enriched with SI semantics, as well as precise timestamps and coordinates of creation.

The behavior of the protocol was evaluated under different application scenarios, and the best configurations were identified. The MAC was highlighted as a limitant aspect of performance, and an analytic model suggests that it is not simply an implementation issue. Possible strategies identified in the literature to employ in a better MAC design include leveraging time synchronization and knowledge about network structure. Other choices of physical layer would also have a significant impact on the performance bottlenecks, as a major part of the MAC's parameters are specifically derived from characteristics of an IEEE 802.15.4 physical layer. As future work, an investigation of the mentioned MAC techniques and a study of alternative physical layers are suggested.

As a more general contribution, it was shown that pigtailing synchronization information to network messages and sharing it across sub-components can lead to better performance than the exchange of explicit synchronization messages. Moreover, a cross-layer architecture to fully leverage information sharing does not necessarily result in a tightly-coupled software design.

As the world becomes more connected, with ever smaller and more pervasive computing devices, efforts towards careful, domain-oriented designs and implementations of novel operating systems and network protocols are important to guide the Internet of Things away from the problems and vulnerabilities identified and accumulated over decades of development in traditional computer networks, so that it can one day drive a smarter, Trustful world.

BIBLIOGRAPHY

AKHAVAN, M. R.; WATTEYNE, T.; AGHVAMI, A. H. Enhancing the performance of RPL using a Receiver-Based MAC protocol in lossy WSNs. In: **IEEE ICT**. Ayia Napa, Cyprus: [s.n.], 2011. p. 191–194.

BACHIR, A. et al. Micro-Frame Preamble MAC for Multihop Wireless Sensor Networks. In: **IEEE ICC**. Istanbul, Turkey: [s.n.], 2006. p. 3365–3370. ISSN 8164-9547.

BERNSTEIN, D. J. The poly1305-aes message authentication code. In: **Proceedings of Fast Software Encryption**. Paris, France: [s.n.], 2005. p. 32–49.

BOULIS, A. **Castalia A simulator for Wireless Sensor Networks and Body Area Networks**. 2017. Available at: <https://github.com/boulis/Castalia>.

CRISTIAN, F. Probabilistic clock synchronization. **Distributed Computing**, Springer, v. 3, n. 3, p. 146–158, set. 1989. ISSN 0178-2770.

DIXON, C. et al. An operating system for the home. In: **Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation**. Berkeley, CA, USA: USENIX Association, 2012. (NSDI'12), p. 25–25. Available at: <http://dl.acm.org/citation.cfm?id=2228298.2228332>.

DJENOURI, D.; BAGAA, M. Synchronization protocols and implementation issues in wireless sensor networks: A review. **IEEE Systems Journal**, Institute of Electrical and Electronics Engineers (IEEE), v. 10, n. 2, p. 617–627, jun 2016. Available at: <https://doi.org/10.1109\%2Fjsyst.2014.2360460>.

DOUKHNITCH, E.; SALAMAH, M.; OZEN, E. An efficient approach for trilateration in 3d positioning. **Computer Communications**, v. 31, n. 17, p. 4124 – 4129, 2008. ISSN 0140-3664. Available at: <http://www.sciencedirect.com/science/article/pii-/S0140366408004751>.

DUNKELS, A. **The ContikiMAC Radio Duty Cycling Protocol**. 2011.

EIDSON, J. C. **Measurement, Control, and Communication Using IEEE 1588 (Advances in Industrial Control)**. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006. ISBN 1846282500.

EL-HOIYDI, A.; DECOTIGNIE, J. D. Wisemac: an ultra low power mac protocol for the downlink of infrastructure wireless sensor networks. In: **Computers and Communications, 2004. Proceedings. ISCC 2004. Ninth International Symposium on**. [S.l.: s.n.], 2004. v. 1, p. 244–251 Vol.1.

FERRARI, F. et al. Efficient network flooding and time synchronization with glossy. In: **Proceedings of the 10th ACM/IEEE International Conference on Information Processing in Sensor Networks**. [S.l.: s.n.], 2011. p. 73–84.

FERRARI, F. et al. The bus goes wireless: Routing-free data collection with qos guarantees in sensor networks. In: **2012 IEEE International Conference on Pervasive Computing and Communications Workshops**. [S.l.: s.n.], 2012. p. 26–31.

FRÖHLICH, A. A.; STEINER, R.; RUFINO, L. M. A trustful infrastructure for the internet of things based on eposmote. In: **9th IEEE International Conference on Dependable, Autonomic and Secure Computing**. [S.l.: s.n.], 2011. p. 63–68.

FU, B. et al. A survey of cross-layer designs in wireless networks. **Communications Surveys Tutorials, IEEE**, v. 16, n. 1, p. 110–126, First 2014. ISSN 1553-877X.

GAMMA, E. et al. **Design Patterns: Elements of Reusable Object-Oriented Software**. [S.l.]: Addison-Wesley, 1995.

GRACIOLI, G. et al. Avaliação de um Algoritmo de Localização baseado em RSSI para Redes Sensores Sem Fio. **Revista IEEE América Latina**, v. 9, n. 1, p. 96–101, 2011. ISSN 1548-0992.

GRANJAL, J.; MONTEIRO, E.; SILVA, J. S. Security for the internet of things: A survey of existing protocols and open research issues. **IEEE Communications Surveys Tutorials**, v. 17, n. 3, p. 1294–1312, thirdquarter 2015. ISSN 1553-877X.

GUSELLA, R.; ZATTI, S. The accuracy of the clock synchronization achieved by tempo in berkeley unix 4.3bsd. **IEEE Transactions on Software Engineering**, v. 15, n. 7, p. 847–853, Jul 1989. ISSN 0098-5589.

HUANG, P. et al. The evolution of mac protocols in wireless sensor networks: A survey. **IEEE Communications Surveys Tutorials**, v. 15, n. 1, p. 101–120, First 2013. ISSN 1553-877X.

HUANG, Q. et al. Fast authenticated key establishment protocols for self-organizing sensor networks. In: **2nd ACM WSNA**. New York, NY, USA: ACM, 2003. (WSNA '03), p. 141–150. ISBN 1-58113-764-8.

HULBERT, A. et al. An experimental study of big spatial data systems. In: **IEEE International Conference on Big Data (Big Data)**. [S.l.: s.n.], 2016. p. 2664–2671.

IEEE. Ieee standard for a precision clock synchronization protocol for networked measurement and control systems. **IEEE Std 1588-2008 (Revision of IEEE Std 1588-2002)**, p. c1–269, July 2008.

IEEE Standard for Local and metropolitan area networks–Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs). **IEEE Std 802.15.4-2011 (Revision of IEEE Std 802.15.4-2006)**, p. 1–314, Sept 2011.

INSTRUMENTATION, I.; SOCIETY", M. **"1451.0 - IEEE Standard for a Smart Transducer Interface for Sensors and Actuators—Common Functions, Communication Protocols, and Transducer Electronic Data Sheet (TEDS) Formats"**. 2007. Online. Available at: <http://web.mit.edu/goretkin/Public/ieee1451/4338161.pdf>.

KARLOF, C.; SASTRY, N.; WAGNER, D. Tinysec: a link layer security architecture for wireless sensor networks. In: **2nd SenSys**. New York, NY, USA: ACM, 2004. p. 162–175. ISBN 1-58113-879-2. Available at: <http://doi.acm.org/10.1145/1031495.1031515>.

LAB, S. I. **EPOS - Embedded Parallel Operating System**. 2017. Available at: <https://epos.lisha.ufsc.br/>.

LEVIS, P. et al. Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks. In: **Proceedings of the 1st Conference on Symposium on Networked Systems Design and Implementation - Volume 1**. Berkeley, CA, USA: USENIX Association, 2004. (NSDI'04), p. 2–2. Available at: <http://dl.acm.org/citation.cfm?id=1251175.1251177>.

LEWIS, L. L. An introduction to frequency standards. **Proceedings of the IEEE**, v. 79, n. 7, p. 927–935, Jul 1991. ISSN 0018-9219.

LISHA. **Internet of Things at UFSC**. 2017. Available at: <https://iot.ufsc.br/>.

LIU, S.; FAN, K. W.; SINHA, P. Cmac: An energy efficient mac layer protocol using convergent packet forwarding for wireless sensor networks. In: **2007 4th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks**. [S.l.: s.n.], 2007. p. 11–20. ISSN 2155-5486.

LONARE, S.; WAHANE, G. A survey on energy efficient routing protocols in wireless sensor network. In: **2013 Fourth International Conference on Computing, Communications and Networking Technologies (ICCCNT)**. IEEE, 2013. Available at: <https://doi.org/10.1109\%2Ficccnt.2013.6726591>.

LU, G.; KRISHNAMACHARI, B.; RAGHAVENDRA, C. S. An adaptive energy-efficient and low-latency mac for data gathering in wireless sensor networks. In: **Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International**. [S.l.: s.n.], 2004. p. 224–.

LUK, M. et al. Minisec: A secure sensor network communication architecture. In: **6th IPSN**. [S.l.: s.n.], 2007. p. 479 –488.

MARóTI, M. et al. The flooding time synchronization protocol. In: **Proceedings of the 2Nd International Conference on Embedded Networked Sensor Systems**. New York, NY, USA: ACM, 2004. (SenSys '04), p. 39–49. ISBN 1-58113-879-2. Available at: <http://doi.acm.org/10.1145/1031495.1031501>.

MEHTA, A. M.; PISTER, K. S. J. Frequency offset compensation for crystal-free 802.15.4 communication. In: **International Conference on Advanced Technologies for Communications (ATC)**. [S.l.: s.n.], 2011. p. 45–47. ISSN 2162-1020.

MENDES, L. D.; RODRIGUES, J. J. A survey on cross-layer solutions for wireless sensor networks. **Journal of Network and Computer Applications**, v. 34, n. 2, p. 523 – 534, 2011. ISSN 1084-8045. Efficient and Robust Security and Services of Wireless Mesh Networks.

MILLS, D. Internet time synchronization: the network time protocol. **IEEE Transactions on Communications**, v. 39, n. 10, p. 1482–1493, Oct 1991. ISSN 0090-6778.

NIAN, M. G. lesta SIVA, J.; POELLABAUER, C. Radio frequency-based indoor localization in ad-hoc networks. In: ORTIZ, J. (Ed.). **Ad Hoc Networks**. InTech, 2017. cap. 6. Available at: <https://www.intechopen.com/books/ad-hoc-networks/radio-frequency-based-indoor-localization-in-ad-hoc-networks>.

OKAZAKI, A. M.; FRÖHLICH, A. A. ADHOP: an Energy Aware Routing Algorithm for Mobile Wireless Sensor Networks. In: **IEEE SENSORS**. Taipei, Taiwan: [s.n.], 2012. Available at: <http://www.lisha.ufsc.br/pub/Okazaki_Sensors_2012.pdf>.

OLIVEIRA, P. et al. Sincronização de Tempo a nível de SO utilizando o protocolo IEEE1588. In: **Brazilian Symposium on Computing Systems Engineering**. Natal, Brazil: [s.n.], 2012. ISBN 978-0-7695-4929-3. Available at: <http://sbesc.lisha.ufsc.br-/sbesc2012/tiki-download_file.php?fileId=115>.

OPENSIM. **OMNeT++ - Objective Modular Network Testbed in C++**. 2017. Available at: <https://omnetpp.org/>.

PATIL, M.; BIRADAR, R. C. A survey on routing protocols in wireless sensor networks. In: IEEE. **Networks (ICON), 2012 18th IEEE International Conference on**. [S.l.], 2012. p. 86–91.

PIRES, R. P.; WANNER, L. F.; FRÖHLICH, A. A. An Efficient Calibration Method for RSSI-based Location Algorithms. In: **6th International IEEE Conference on Industrial Informatics**. Daejeon, Korea: [s.n.], 2008. p. 1183–1188. ISBN 978-1-4244-2170-1.

POLASTRE, J.; HILL, J.; CULLER, D. Versatile low power media access for wireless sensor networks. In: **ACM SenSys**. New York, USA: [s.n.], 2004. p. 95–107. ISBN 1-58113-879-2.

REGHELIN, R.; FRÖHLICH, A. A. A Decentralized Location System for Sensor Networks Using Cooperative Calibration and Heuristics. In: **9th ACM/IEEE International Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems**. Torremolinos, Malaga, Spain.: [s.n.], 2006. p. 139–146. ISBN 1-59593-477-4.

RESNER, D. **Estabelecimento de Chaves e Comunicação Segura para a Internet das Coisas**. Florianópolis: [s.n.], 2014. 67 p. B.Sc. Thesis. Available at: <http://www.lisha.ufsc.br/pub/Resner_BSC_2014.pdf>.

RESNER, D.; ARAUJO, G. M. de; FRÖHLICH, A. A. On the Impact of Dynamic Routing Metrics on a Geographic Protocol for WSNs. In: **Brazilian Symposium on Computing Systems Engineering.** João Pessoa, Brazil: [s.n.], 2016. Available at: <http://www.lisha.ufsc.br/pub/Resner_SBESC_2016.pdf>.

RESNER, D.; ARAUJO, G. M. de; FRÖHLICH, A. A. Design and Implementation of a Cross-Layer IoT Protocol. **Science of Computer Programming**, v. 0, n. 0, p. 0, 2017. ISSN 0000-0000.

RESNER, D.; FRÖHLICH, A. A. Design Rationale of a Cross-layer, Trustful Space-Time Protocol for Wireless Sensor Networks. In: **20th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA).** Luxembourg, Luxembourg: [s.n.], 2015. p. 1–8. Available at: <http://www.lisha.ufsc.br/pub/Resner_ETFA_2015.pdf>.

RESNER, D.; FRÖHLICH, A. A. Key Establishment and Trustful Communication for the Internet of Things. In: **4th International Conference on Sensor Networks (SENSORNETS 2015)**. Angers, France: [s.n.], 2015. p. 197–206. ISBN 978-989-758-086-4. Available at: <http://www.lisha.ufsc.br/pub/Resner_SENSORNETS_2015.pdf>.

RESNER, D.; FRÖHLICH, A. A. TSTP MAC: A Foundation for the Trustful Space-Time Protocol. In: **14th IEEE/IFIP International Conference on Embedded and Ubiquitous Computing (EUC).** Paris, France: [s.n.], 2016.

RESNER, D.; FRÖHLICH, A. A.; WANNER, L. F. Speculative Precision Time Protocol: submicrosecond clock synchronization for the IoT. In: **21th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)**. Berlin, Germany: [s.n.], 2016.

SANTOS, T. R. dos; FRÖHLICH, A. A. A Customizable Component for Low-Level Communication Software. In: **19th Annual Symposium on High Performance Computing Systems and Applications**. Guelph, Canada: [s.n.], 2005. p. 58–64. ISBN 0-7695-2343-9.

SCHMID, T.; DUTTA, P.; SRIVASTAVA, M. B. High-resolution, low-power time synchronization an oxymoron no more. In: **Proceedings of the 9th ACM/IEEE International Conference**

on Information Processing in Sensor Networks. New York, NY, USA: ACM, 2010. (IPSN '10), p. 151–161. ISBN 978-1-60558-988-6. Available at: <http://doi.acm.org/10.1145/1791212.1791231>.

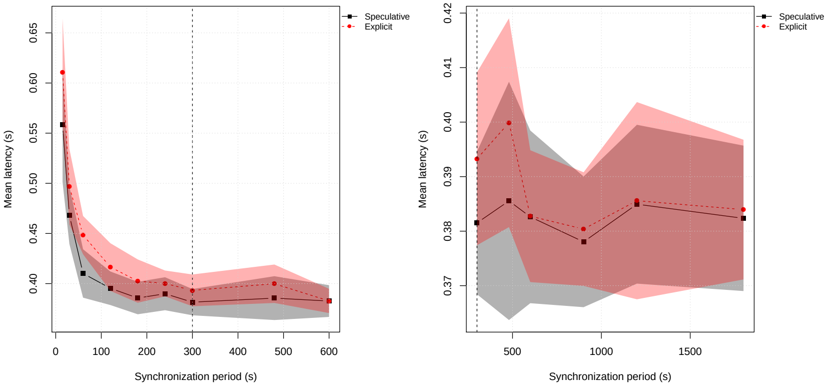
STEINER, R. V. et al. Performance Evaluation of Receiver Based MAC Using Configurable Framework in WSNs. In: **IEEE Wireless Communications and Networking Conference (WCNC)**. Shanghai, China: [s.n.], 2013. p. 884–889. ISBN 978-1-4673-5939-9. Available at: <http://www.lisha.ufsc.br/pub/Steiner_WCNC_2013.pdf>.

SUO, H. et al. Security in the internet of things: A review. In: **ICCSEE**. [S.l.: s.n.], 2012. v. 3, p. 648–651.

TEXAS INSTRUMENTS. **CC2538 Powerful Wireless Microcontroller System-On-Chip for 2.4-GHz IEEE 802.15.4, 6LoWPAN, and ZigBee Applications datasheet**. [S.l.], 4 2015. Rev. D.

ZHOU, H. et al. **Frequency Accuracy and Stability Dependencies of Crystal Oscillators**. [S.l.], 11 2008.

Appendix A – Additional simulation results



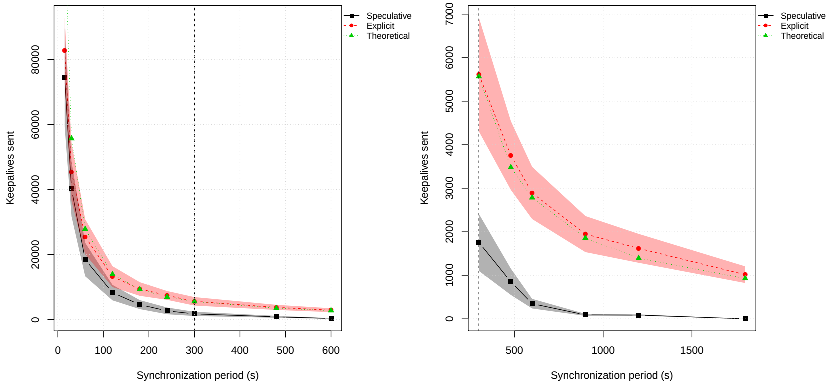
(a) Low synchronization periods

(b) High synchronization periods

Figure 96: Average latency for environment monitoring scenario, d=300s.

The simulation results in this Appendix are similar to the ones presented and discussed in Chapter 4, and are included for completion.

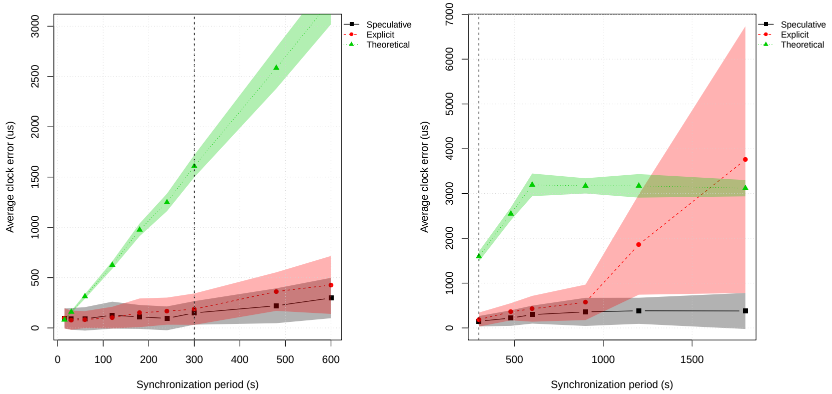
A.1 SYNCHRONIZATION



(a) Low synchronization periods

(b) High synchronization periods

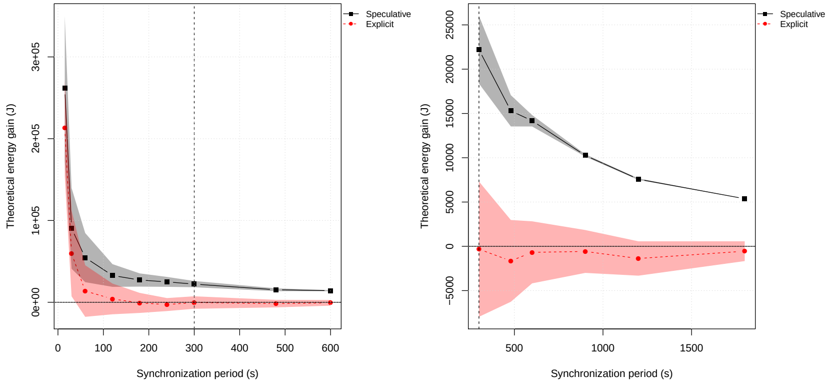
Figure 97: Number of explicit synchronization messages for environment monitoring scenario, d=300s.



(a) Low synchronization periods

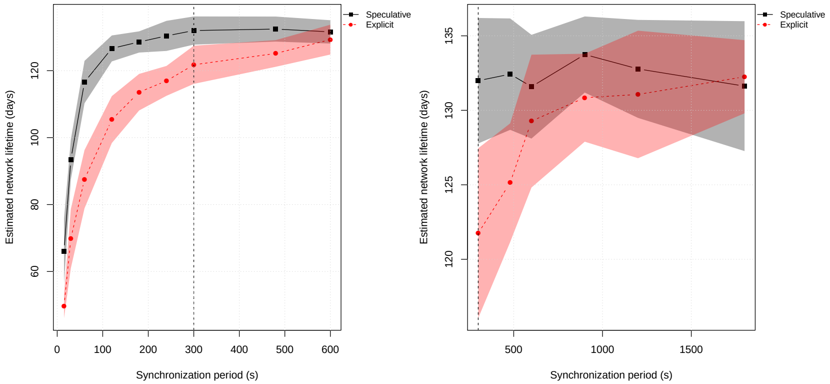
(b) High synchronization periods

Figure 98: Average clock error for environment monitoring scenario, d=300s.



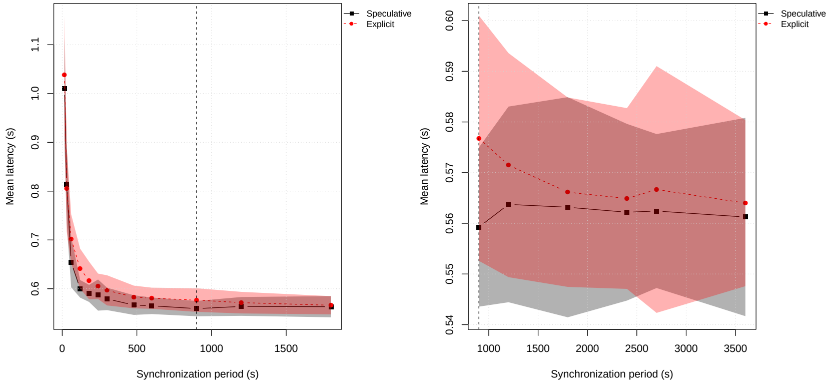
(a) Low synchronization periods (b) High synchronization periods

Figure 99: Energy consumption compared to theoretical explicit approach for environment monitoring scenario, d=300s.



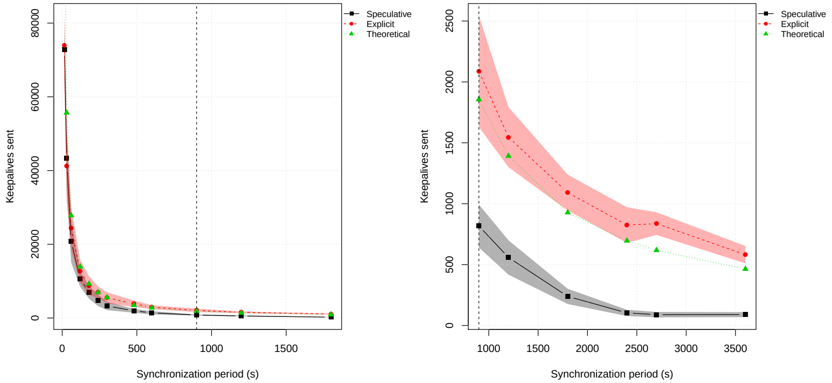
(a) Low synchronization periods (b) High synchronization periods

Figure 100: Estimated network lifetime for environment monitoring scenario, d=300s.



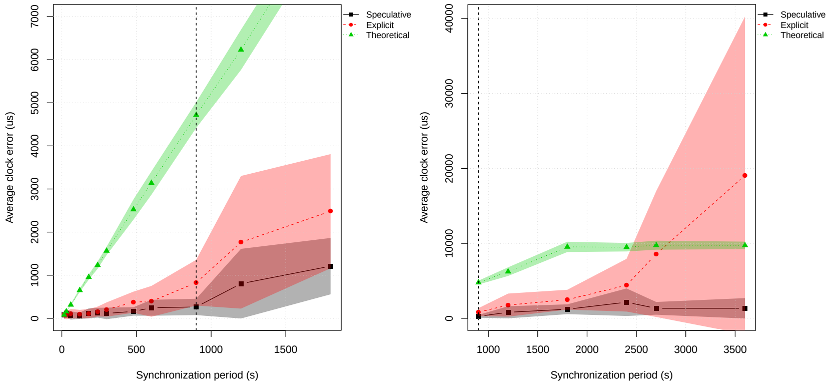
(a) Low synchronization periods (b) High synchronization periods

Figure 101: Average latency for environment monitoring scenario, d=900s.



(a) Low synchronization periods (b) High synchronization periods

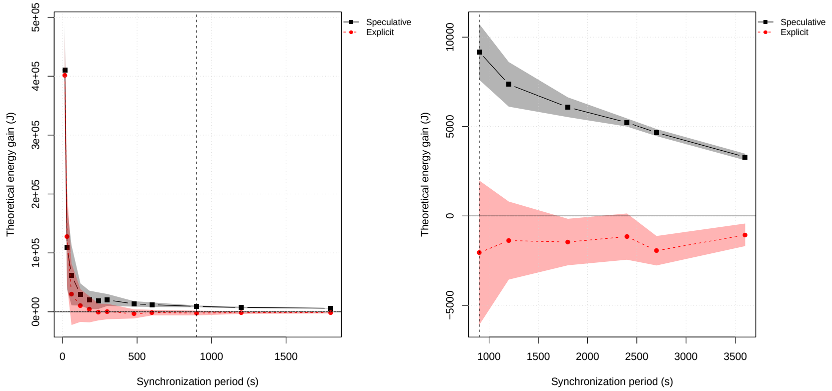
Figure 102: Number of explicit synchronization messages for environment monitoring scenario, d=900s.



(a) Low synchronization periods

(b) High synchronization periods

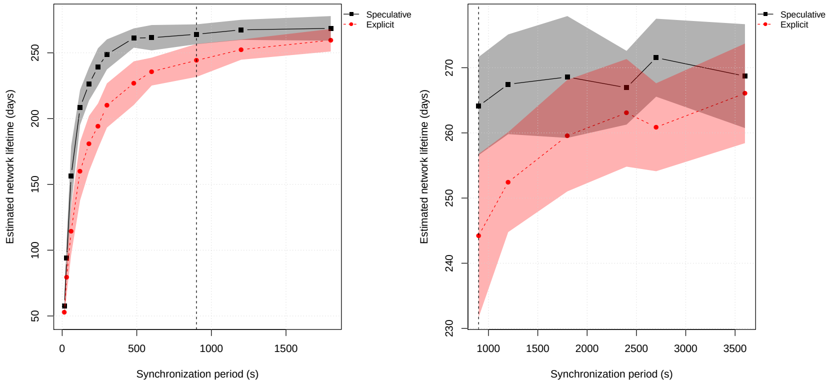
Figure 103: Average clock error for environment monitoring scenario, d=900s.



(a) Low synchronization periods

(b) High synchronization periods

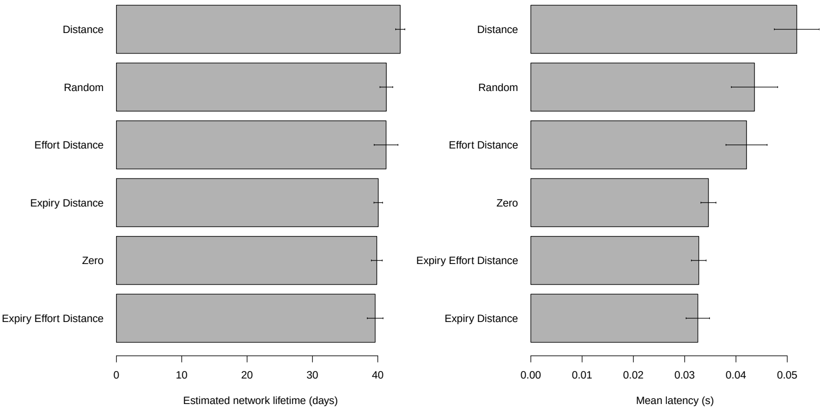
Figure 104: Energy consumption compared to theoretical explicit approach for environment monitoring scenario, d=900s.



(a) Low synchronization periods

(b) High synchronization periods

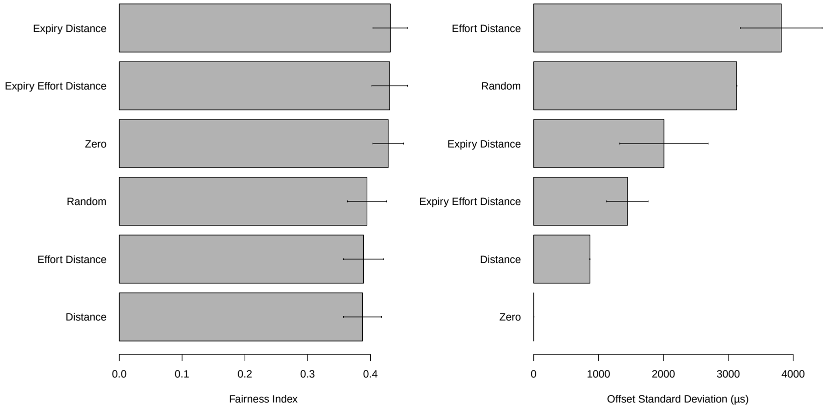
Figure 105: Estimated network lifetime for environment monitoring scenario, d=900s.



(a) Network lifetime (b) Mean latency

Figure 106: Estimated network lifetime and mean latency for LISHA office scenario under different routing metrics.

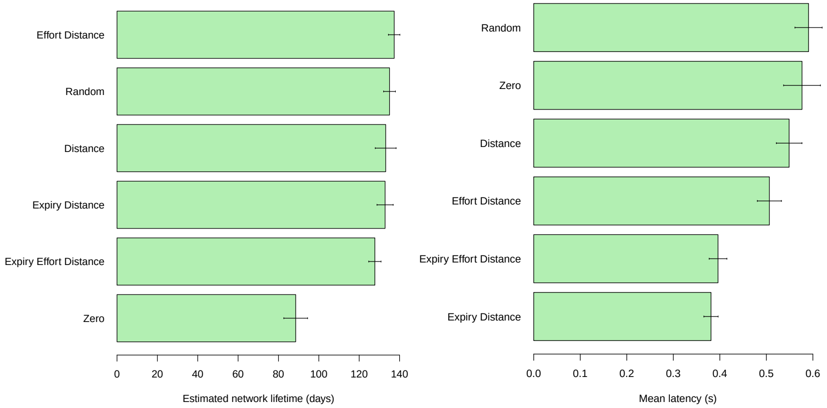
A.2 ROUTING METRICS



(a) Faienss index

(b) Offset standard deviation

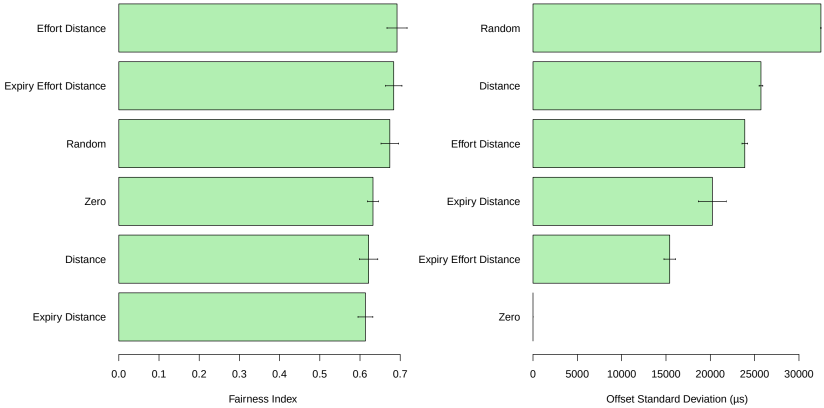
Figure 107: Fairness index and offset standard deviation for LISHA office scenario under different routing metrics.



(a) Network lifetime

(b) Mean latency

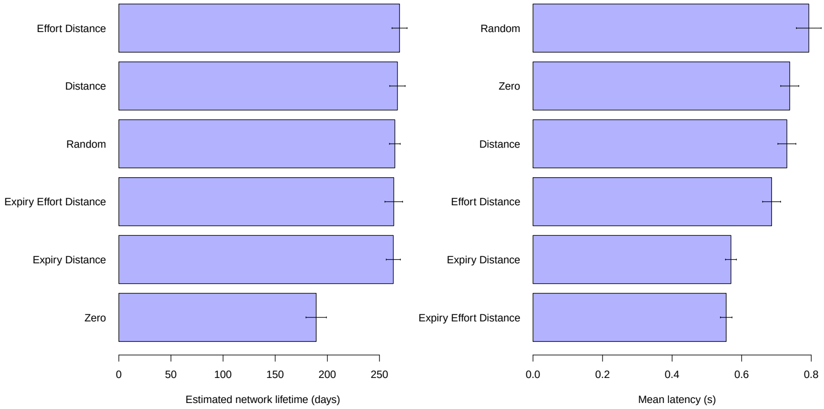
Figure 108: Estimated network lifetime and mean latency for environment monitoring scenario, d=300s, under different routing metrics.



(a) Fainess index

(b) Offset standard deviation

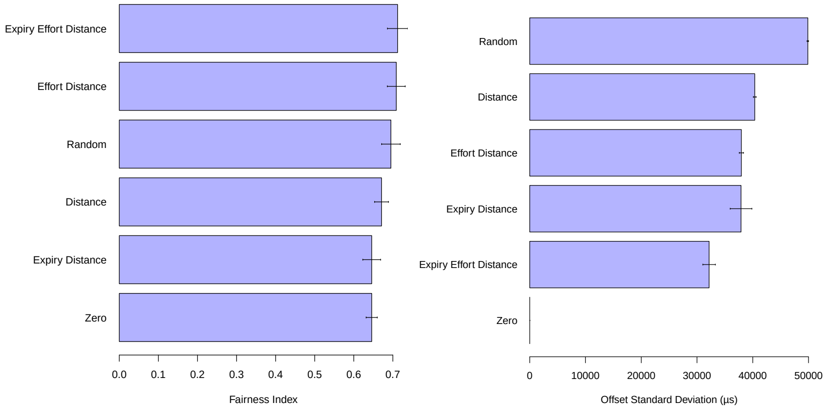
Figure 109: Fairness index and offset standard deviation for environment monitoring scenario, d=300s, under different routing metrics.



(a) Network lifetime

(b) Mean latency

Figure 110: Estimated network lifetime and mean latency for environment monitoring scenario, d=900s, under different routing metrics.



(a) Fainess index

(b) Offset standard deviation

Figure 111: Fairness index and offset standard deviation for environment monitoring scenario, d=900s, under different routing metrics.

Appendix B – Scientific Publications

The following articles were direct products of the work presented in this dissertation, published in scientific journals or conference proceedings:

1. RESNER, D.; FROHLICH, A. A. . **Key Establishment and Trustful Communication for the Internet of Things.** In: 4th International Conference on Sensor Networks (SENSORNETS), 2015, ESEO. p. 197-206.

2. RESNER, D.; FROHLICH, A. A. . **Design rationale of a cross-layer, Trustful Space-Time Protocol for Wireless Sensor Networks.** In: 2015 IEEE 20th Conference on Emerging Technologies & Factory Automation (ETFA), 2015, Luxembourg.

3. RESNER, D.; FROHLICH, A. A. . **TSTP MAC: a Cross-Layer, Geographic, Receiver-Based MAC Protocol for WSNs.** In: Brazilian Symposium on Computing Systems Engineering (SBESC), 2015, Foz do Iguaçu.

4. RESNER, D.; FROHLICH, A. A. . **TSTP MAC: A Foundation for the Trustful Space-Time Protocol.** In: 2016 IEEE Intl Conference on Embedded and Ubiquitous Computing (EUC), Paris. 2016, p. 40.

5. RESNER, D.; FROHLICH, A. A.; WANNER, L. F. . **Speculative Precision Time Protocol: Submicrosecond clock synchronization for the IoT.** In: 2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA), 2016, Berlin. 2016.

6. RESNER, D.; ARAUJO, G. M. ; FROHLICH, A. A. . **On the Impact of Dynamic Routing Metrics on a Geographic Protocol for WSNs.** In: 2016 VI Brazilian Symposium on Computing Systems Engineering (SBESC), 2016, João Pessoa. p. 109.

7. SILVA, D. S. ; RESNER, D. ; SOUZA, R. L. ; MARTINA, J. E. . **Formal Verification of a Cross-Layer, Trustful Space-Time Protocol for Wireless Sensor Networks.** In: Lecture Notes in Computer Science. 0ed.: Springer International Publishing, 2016, v. , p. 426-443.

8. SUSIN, M. M. ; WANNER, L. F. ; RESNER, D. ; FROHLICH, A. A. . **Time Synchronization under Temperature and**

Distance Variations. In: 2017 VII Brazilian Symposium on Computing Systems Engineering (SBESC), 2017, Curitiba. p. 167.

9. RESNER, D.; ARAUJO, G. M. ; FRÖHLICH, A. A. . **Design and Implementation of a Cross-Layer IoT Protocol.** In: Science Of Computer Programming, 2017.