UNIVERSIDADE DE LISBOA
Faculdade de Ciências
Departamento de Informática

SECURE GPS CLOCK SYNCHRONIZATION IN SMART GRIDS

Radu Petruţ Onica

Dissertação orientada pelo Prof. Doutor Nuno Fuentecilla Maia Ferreira Neves
e co-orientado pelo Prof. Doutor António Casimiro Ferreira da Costa

DISSERTATION

MESTRADO EM SEGURANÇA INFORMÁTICA

2015

Acknowledgements

Firstly, I would like to express my sincere gratitude to my advisor Prof. Nuno Neves as well as my co-advisor Prof. António Casimiro for the continuous support of my Masters thesis, for their patience and motivation.

I am sincerely grateful to Rodrigo, Soraia and Ricardo for all their support during the writing of this dissertation as well as the motivation they gave me, to help me overcome the hardest obstacles I faced. A huge thanks to all my colleagues, Bernardo, Pedro, Miguel, Tiago, Maneca and André for the all the memorable moments and all the fun we had this past year.

Lastly I would like to thank my family and especially my sister for always encouraging me and giving me motivation to move forward both throughout this thesis and my life in general.

In memory of my Grandmother.

Resumo

As smart grids resultaram da integração da rede elétrica atual no mundo digital. Isso traz várias vantagens às redes elétricas, como uma instalação, configuração e manutenção mais simples e eficiente, mas também a fácil integração na rede de novas tecnologias. Enquanto as redes elétricas continuam a crescer em dimensão e complexidade, elas tornam-se mais importantes para a sociedade e subsequentemente mais sujeitas a ataques distintos. Alguns dos objetivos mais importantes da smart grid são: acomodar uma grande variedade de tecnologias de produção de eletricidade como a eólica, solar e geotérmica; ser resiliente a ataques físicos e ciber-ataques; ter mecanismos de deteção, análise e resposta automática a incidentes; dar mais poder ao consumidor final sobre como e quando a energia pode ser comprada ou consumida.

Para implementar actividades relacionadas com a monitorização do estado da smart grid, vários componentes especializados são geograficamente distribuídos pela rede. Um dos dispositivos críticos é o Phase Measurement Unit (Unidade de Medição de Fase) (PMU). Este dispositivo é usado para estimar o estado da smart grid num determinado momento, recolhendo várias métricas sobre a qualidade do sinal elétrico. Para se conseguir criar uma imagem geral da rede inteira, todos estes dispositivos necessitam de ser sincronizados no tempo, assegurando assim que as medições são efetuadas aproximadamente no mesmo instante.

A sincronização do tempo desempenha um papel crucial na estabilidade e no funcionamento correto de todos os componentes da smart grid. Dada a importância da sincronização de tempo, e a falta de qualquer tipo de proteção nas soluções atuais, este sistema torna-se num alvo potencial para atacantes.

Em conformidade com os standards, a precisão dos relógios dos PMU's devem ter um erro máximo na ordem dos 30 μs. Isso garante que a informação recolhida sobre o estado da smart grid é válida. Hoje em dia este requisito é satisfeito usando equipamentos GPS em cada sítio onde se encontra um PMU. Quando o GPS foi concebido, não se pensou que podia vir a ter o sucesso e o impacto atual e, portanto, assegurar a sua segurança não foi um ponto importante. Ao longo do tempo passou a ser usado em infraestruturas críticas, o que introduz eventuais problemas graves de segurança. As smart grids são uma destas estruturas críticas onde o GPS está a ser usado sem qualquer tipo de proteção. Atualmente existe também uma versão segura do GPS que é empregue pelas forças militares. Os

dispositivos que conseguem decifrar este sinal só estão disponíveis ao exército. Por além disso, todos os detalhes sobre o funcionamento do algoritmo de cifra são mantidos em segredo.

Ao longo dos anos foram desenvolvidos vários tipos de ataques ao GPS. O mais básico é o *Blocking* que consiste simplesmente em impedir a comunicação entre a antena do recetor e o sinal GPS. Isso pode ser conseguido de uma maneira tão simples como tapar a antena com um bocado de metal. Um ataque que tenta também quebrar a ligação com o satélite é o *Jamming*. A ideia deste ataque é introduzir ruído suficiente para que o recetor não consiga distinguir o sinal original. Estes dois tipos de ataques só conseguem perturbar o funcionamento do recetor GPS. Um tipo de ataque mais potente é o *Spoofing*. Este ataque consegue modificar o sinal original vindo do satélite de forma a enganar o recetor. Assim é possível fazer com que o recetor GPS mostre uma posição ou tempo incorretos. Nesta dissertação também foi analisada uma evolução deste ataque que tem como alvo a alteração ilegítima dos dados contidos no sinal. Isso pode fazer como que o recetor falhe ou deixe de poder ser usado.

Os algoritmos de sincronização de relógios existentes hoje em dia, nomeadamente o *Network Time Protocol (NTP)* e o *Precision Time Protocol (PTP)*, não são suficientemente robustos, em termos de segurança ou precisão, para serem utilizados na smart grid. O NTP foi concebido para a sincronização de relógios em redes de grande escala mas não consegue fornecer a precisão necessária para os requisitos da smart grid. Por outro lado temos o PTP que consegue atingir uma precisão na ordem dos nanosegundos em certas condições, mas é muito sensível a atrasos e oscilações na rede. Isso faz com que o PTP só consiga garantir uma precisão de tempo na ordem dos nanosegundos em redes de pequena escala. A smart grid usa uma rede de alta velocidade com relativamente pouco tráfego, o que torna o PTP uma possível solução para algumas partes dessa rede. Em termos de segurança, o PTP não está preparado para ser utilizado num ambiente tão crítico como a smart grid, sendo suscetível a ataques.

O foco desta investigação é encontrar um algoritmo resiliente a faltas, capaz de satisfazer os requisitos de sincronização de tempo necessários para o correto funcionamento da smart grid. Foi desenvolvida uma solução baseada no PTP, que consegue cumprir os requisitos de precisão temporal na smart grid e também consegue mitigar todos os tipos de ataques ao GPS que foram identificados. Para além disso, a solução também permite reduzir o número de recetores de GPS necessários para o funcionamento correto da smart grid.

Palavras-chave: Smart Grid, GPS, Sincronização de relogios, Segurança

Abstract

Smart grids resulted from the integration of computer technologies into the current power grid. This brings several advantages, allowing for a faster and more efficient deployment, configuration and maintenance, as well as easy integration of new energy sources (e.g., wind and solar). As smart grids continue to grow in size and complexity, they become subject to failures and attacks from different sources. Time synchronization plays a crucial role in the stability and correct functioning of many grid components. Considering how sensitive time synchronization is, the tight restrictions imposed for correct operation and the lack of any kind of protection, makes this service a potential prime target for attackers. Today most of the time synchronization requirements are met using relatively expensive GPS hardware placed in some locations of the smart grid. When GPS was first devised, nobody could have predicted the success and the impact that it would have and therefore, security was never an important concern. Through the years, it slowly gained entrance into more critical systems, where it was never intended to be used, which can lead to serious security problems. The smart grid is just one of these critical systems where GPS is being employed without any kind of protection. The focus of this research is trying to solve this problem, by proposing a more secure and robust clock synchronization algorithm. A solution based on the Precision Time Protocol (PTP) was developed that manages to fulfill the time synchronization requirements of the smart grid and is also capable of mitigating all types of identified GPS attacks. As an added benefit, the solution may also reduce the number of GPS receivers necessary for the correct operation of the smart grid, contributing to decrease costs.

Keywords: Smart Grid, GPS, Time synchronization, Security

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The smart grid represents the evolution and integration into the digital world of the current electrical grid. Information technologies are used to improve the way the electric generation and distribution is currently done, making it more efficient, reliable and sustainable. The smart grid will support better automation, management and coordination between the end-user and the producer. Advantages include advanced monitoring that can accurately identify and take measures to mitigate failures and attacks. Self-healing technologies can be implemented that will allow the grid to solve problems without the need for on-site verifications or any kind of human intervention.

This increase in interaction between production and consumption, as well as the sheer complexity demanded by such a network, leads to a set of new challenges and security problems. User privacy becomes a real concern because of the increase in the data collection and processing. Reliable communication among the various parts of the electrical infrastructure turns into a critical necessity.

Smart grids are composed of different types of components each with its own requirements. One such component is the Phase Measurement Unit (PMU), which is an advanced measurement device used to obtain status information about the electrical lines. It is one of the building blocks of a monitoring system that can analyze the state of the grid in real-time. In particular, a PMU keeps track of various parameters (e.g., electrical current amplitude and phase) of the part of the smart grid where it is installed. This can be thought of as a ”snapshot” of the state in which that portion of the smart grid is in. The deployment of multiple PMU’s at different locations across the network allows for a complete state estimation. This information can then be used to support decisions on the need to modify the power generation or the distribution of loads, which in turn gives

the ability to predict and avoid failures like blackouts. Time synchronization among all PMU's is critical to create a coherent picture of the smart grid. If PMU's take "snapshots" at different times, the complete state cannot be correctly estimated.

The Global Positioning System (GPS) is used to synchronize the clocks of the PMU's around the network [2]. GPS is a type of satellite based clock synchronization solution. It is able to provide time reference with sufficient accuracy and availability to be used in infrastructures and systems where time synchronization is critical. When it was first designed, security was not a concern. However, as it grew in popularity, this lack of security became a more troubling concern.

One way to try to address the lack of security of GPS is to use a Software Defined Radio (SDR) [3] capable receiver and modify it in such a way to make it more resilient to attacks. SDR as a concept has been around for some years now, but recently has experienced a rapid evolution, taking advantage of the newer capabilities of digital electronics. SDR allows for a software implementation of several components that traditionally were only possible in hardware, like radio filters, modulators/demodulators and detectors. This not only reduces costs and increases ease of use, but can also create better open source alternatives to current offerings. Even a full GPS receiver can be implemented in software [4] [Figure 1.1]. This would allow for the addition of features to the standard GPS protocol to make it more secure and suitable for use in smart grids.

Figure 1.1: SDR based GPS receiver.

Another approach to improve security takes advantage of alternative network time synchronization mechanisms. These solutions are advancing at a fast pace and they are currently being studied as a means to synchronize clocks in critical infrastructures, like the electrical grid [5]. Several synchronization systems exist, differing from each other in terms of features, complexity and performance. Today, the most common time synchronization algorithms are the Network Time Protocol (NTP) [6] and the Precision Time

Protocol (PTP) [7].

NTP is a network protocol employed to synchronize system clocks among a set of distributed time servers and clients. Since its first appearance in 1985, four versions were released, with the last one being NTPv4 in 2010 [8]. NTPv4 addresses many of NTPv3's shortcomings, and it can achieve an accuracy of a few milliseconds over the internet, while coping relatively well with variable message delays and dropped packets. It uses a client-server model [see Figure 1.2] with participants organized into different levels called Stratum. Each stratum has a number (starting from 0 at the highest level) representing its distance from the reference clock. The largest stratum level is 15. After this level all nodes are considered desynchronized. Note that even though it seems logical that higher-level clocks have better accuracy, this is not always true because the stratum level can be user-defined and as such it does not always reflect the clock quality.

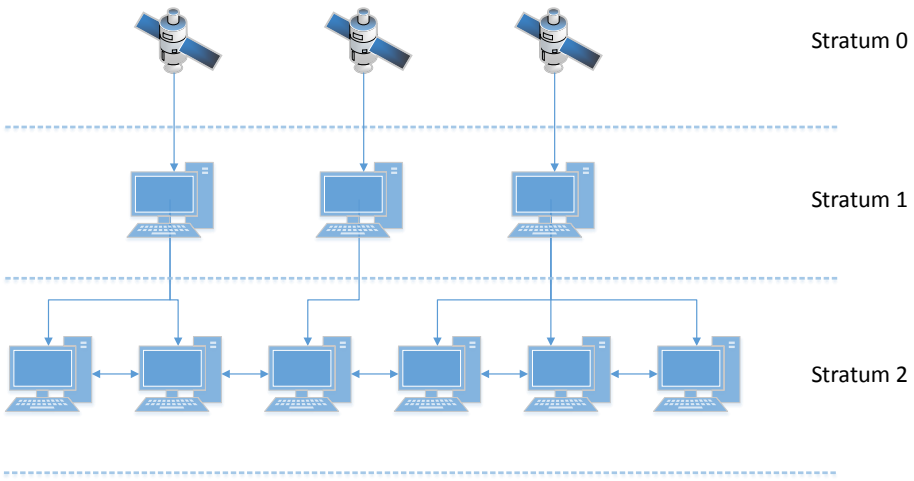

Figure 1.2: Simple NTP Network.

PTP is currently the most accurate network time synchronization protocol available, with an accuracy that can reach nanosecond precision. It makes use of a hierarchical master-slave architecture to synchronize clocks in a network. Different roles are defined for every clock/node but also to each communication link between them. Special messages that carry very accurate timestamps are exchanged between nodes. In order to reach such high levels of synchronization special care needs to be taken with these timestamps. Even the delays introduced by the packet traversing the Operating System (OS) protocol stack can have a big impact on the outcome. To address these difficulties, PTP defines three modes of operation: software-only, hardware-assisted and hardware only. The software-only is the least precise approach because all timestamps are taken at the application level of the protocol stack. On the other hand, hardware-assisted and hardware-only

modes use special hardware to timestamp the messages as close to the physical layer as possible. The difference between these two modes is related to the place where the PTP implementation actually runs - in software for the hardware-assisted mode and in hardware for the hardware-only mode.

1.1 Requirements

The requirements of a time synchronization protocol suitable for use in smart grids are explained in detail in the standard C37.118-2011 [9]. The standard is split into two parts: the first one presents the measurement (called phasor in this context) estimation requirements while the second one explains the network communication protocol. The time synchronization requirements are described as the maximum allowed error for PMU's. Called the Total Vector Error (TVE), it represents the vectorial difference between the estimated and the theoretical phasors in percentages. The requirement imposed by the standard is a maximum of 1% TVE, which equates to approximately 30μs. Even though the standard does not talk about security requirements, the protocol needs to be relatively secure and maintain timing errors below the thresholds. Ideally, this should occur even in extreme situations, such as in the case of an attack or when fault tolerance mechanisms need to do recovery. The fact that each PMU is equipped with a GPS receiver means that it also inherits all the insecurity and problems related to GPS, which in turn makes the implementation of a secure time synchronization algorithm even more challenging.

1.2 Contributions

In this thesis we propose an alternative way of keeping the PMU's synchronized by resorting to a modified version of the PTP protocol. In failure/attack free intervals, the PMU's continue to employ GPS based clock synchronization. When a problem is detected they switch to our protocol.

The changes to PTP were designed with the following two goals:

- the first aim was to create a protocol that manages to satisfy the time synchronization requirements of the smart grid, while also being able to cope with the attacks and failures that might happen around the network;

- as a second objective, we wanted to minimize the changes to the PTP standard, as this facilitates the updates to the existing implementations and the security evaluation.

Most security problems related to PTP are analyzed in some detail in the thesis. There is however a focus on a critical attack vector, namely the GPS receiver, because it functions as the time source for the network devices. As a result of this work, a publication was accepted in the security track of the INFORUM conference.

R. Onica, N. Neves, and A. Casimiro, "Fault-Tolerant Precision Time Protocol for Smart Grids," in Proceedings of 7º Simpósio Nacional de Informática INFORUM, September 2015

1.3 Timeline

The initial work plan for the thesis is presented as a Gantt chart in [Figure 1.3]. All work tasks are described by their respective ID's (first column of the chart).

ID	Start	Finish	Duration
1	07-10-2014	11/28/2014	39d
2	27-10-2014	11/28/2014	25d
3	03-11-2014	12/26/2014	40d
4	08-12-2014	1/30/2015	40d
5	19-01-2015	5/1/2015	75d
6	12-02-2015	7/7/2015	104d

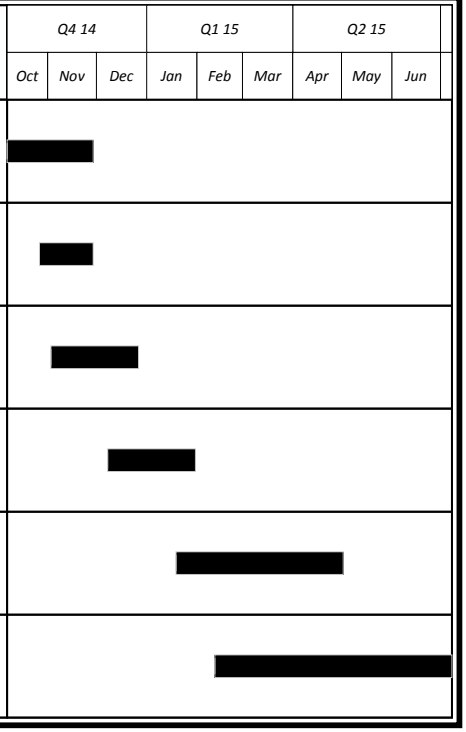

Figure 1.3: Gantt chart representing the initial thesis work plan.

- **Task 1** consists of the study of current GPS and network time synchronization algorithms, with a focus being put on their security, or lack thereof. The NTP and PTP protocol were studied and possible solutions to their respective security problems were researched. The environment in which these protocols would need to work, namely the smart grids, was analyzed to better understand the exact requirements;

- **Task 2** is the design of an approach that could eliminate (or at least diminish) the impact of possible attacks on the time synchronization. The first approach we came up with was based on using SDR to add security features to the current GPS protocol;

- **Task 3** is the analysis of all available hardware and software solutions to select the ones that would allow for the creation of a framework and workbench to be used for implementing, testing and assessing our proposed approach;

- **Task 4** consists of building the framework and workbench with the selected hardware and software;

- **Task 5** is the implementation of the approach and the assessment of its effectiveness. The implementation should be tested against attacks that could happen in the context of a smart grid. The results should be analyzed and compared with the initial requirements;

- **Task 6** is to further analyze how the solution performs and to document the results.

The initial approach was based on using SDR along with specific hardware to try to add protection mechanisms to the standard GPS protocol. SDR is responsible for creating the navigation and time solutions from the RAW GPS data that the hardware provides. However, once we reached **Task 4**, we encountered several compatibility issues with the hardware that lead to unexpected delays in the work plan. When all compatibility issues were resolved, we already had a working prototype of the software that could be tested. At that time, we ran into a second difficulty - the hardware was unable to capture any kind of GPS data.

The problem appeared to be with the SDR dongle or with the antenna - either our SDR dongle was faulty and could not read any kind of GPS data or the antenna was not sufficiently powerful to capture the satellite signal. The dongle managed to collect radio waves in the normal frequency spectrum (like radio stations), which we could decode, but could not receive any kind of signal in the 1575 MHz range that the GPS uses. This

led us to think that the problem was with the antenna. To try to find a solution, we experimented with another antenna of a higher quality and additional hardware, namely a bias-T network to power the new antenna. Even so, we had no success at receiving any kind of GPS signal.

This led us to try a different approach. This was the solution that we ended up implementing and testing. It is based on the PTP, and due to time constraints, we decided to employ a software-only version based on the open-source project ptpd2 [10]. Setting up this solution only required a few Linux PC's and an additional Ethernet card. The first stage was to design the solution to the smallest detail. The next step was to set up the workbench and install the ptpd2 software. After this, the modifications to the standard protocol were implemented and tested.

ID	Start	Finish	Duration
1	10/7/2014	11/28/2014	39d
2	10/27/2014	11/28/2014	25d
3	11/3/2014	12/26/2014	40d
4	12/8/2014	1/30/2015	40d
5	1/19/2015	2/27/2015	30d
6	3/2/2015	4/2/2015	24d
7	4/3/2015	7/7/2015	68d
8	4/3/2015	7/7/2015	68d

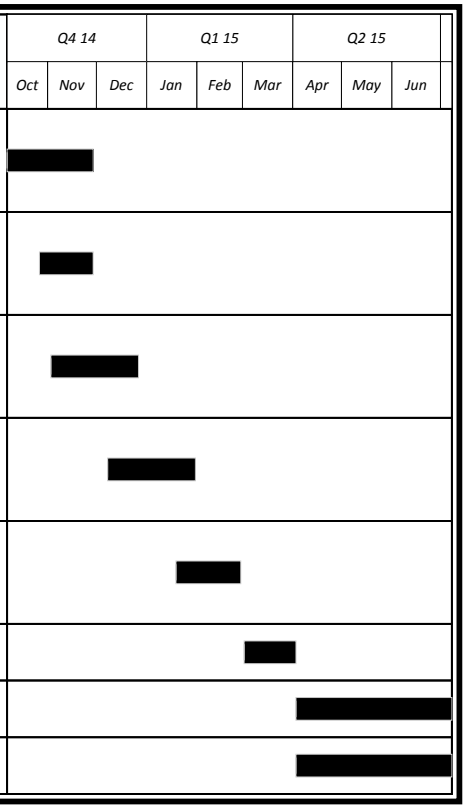

Figure 1.4: Gantt chart representing the final thesis work plan.

The new Gantt chart representing the final work plan for the thesis is shown in [Figure 1.4]. Up to **Task 4** the plan stayed the same but, the final stages of the work had to be revised:

- **Task 5** represents a new research stage in which a different approach was studied to try to adapt the PTP standard in such a way to be used in smart grids;

- **Task 6** corresponds to the design of a new solution that is suited for use in smart grids and can satisfy all the requirements. This task also includes building the workbench for implementing and testing the protocol;
- **Task 7** represents the implementation of the newly designed solution. The implementation is an extension to the PTP standard programmed in the ptpd2 software;
- **Task 8** is the testing phase and also contains the documentation of all the work up until this point.

1.4 Thesis outline

In chapter 2, the concept of smart grid is introduced alongside the best time synchronization protocols, GPS and PTP. In particular, it explains why the PMU's deployed in smart grids need to be time synchronized. After explaining the basics about how GPS and PTP work, the chapter discusses several issues that these protocols have, mainly concerning their security. For PTP, the major problems in terms of time synchronization are described, namely clock offset and delay asymmetry.

Chapter 3 of the thesis is dedicated to give an in depth explanation of the proposed solution the implementation. After presenting the reasoning behind some of the security related decisions, the attack model explains the capabilities of the adversary. All the changes made to the PTP standard are explained next. The last part of this chapter is dedicated to explaining the details of implementing the attack detection mechanisms using the software of ptpd2.

Chapter 4 presents the tests and experimental results of the work. A detailed explanation is given about the testing platform. The implications and effects of using the existing hardware as the test bed are analyzed and various types of attacks are conducted. The result shows the effectiveness of the proposed solution, making it a suitable candidate for the smart grid time synchronization.

Chapter 2

Context

A thorough study of all the current time synchronization algorithms is essential to lay the foundation for our proposed solution. Understanding the challenges imposed by the smart grid provide motivation and give a clear explanation of what the solution is supposed to do. In this chapter the foundation for this thesis is explained. Some related work on the security of currently available solutions is also discussed.

2.1 The Smart Grid

The smart grid [Figure 2.1] is the next generation of the electrical power system. The smart grid has two primary features: it allows flexible and real-time decision making based on real time data collection and analysis; and it is capable of monitoring its own health and alert operators immediately when problems arise. In some cases, the smart grid may also act automatically, taking corrective actions that will minimize the impact of a failure.

Some of the most important objectives of the smart grid are:

- Accommodate a large variety of energy production technologies including wind, solar and geothermal heat;
- Engage the consumer not only in the consumption cycle but also in the energy production cycle;
- Resilience to attacks, as it needs to mitigate both physical and cyber-attacks;
- Self-healing, since it needs to rapidly detect, analyze and respond to incidents;

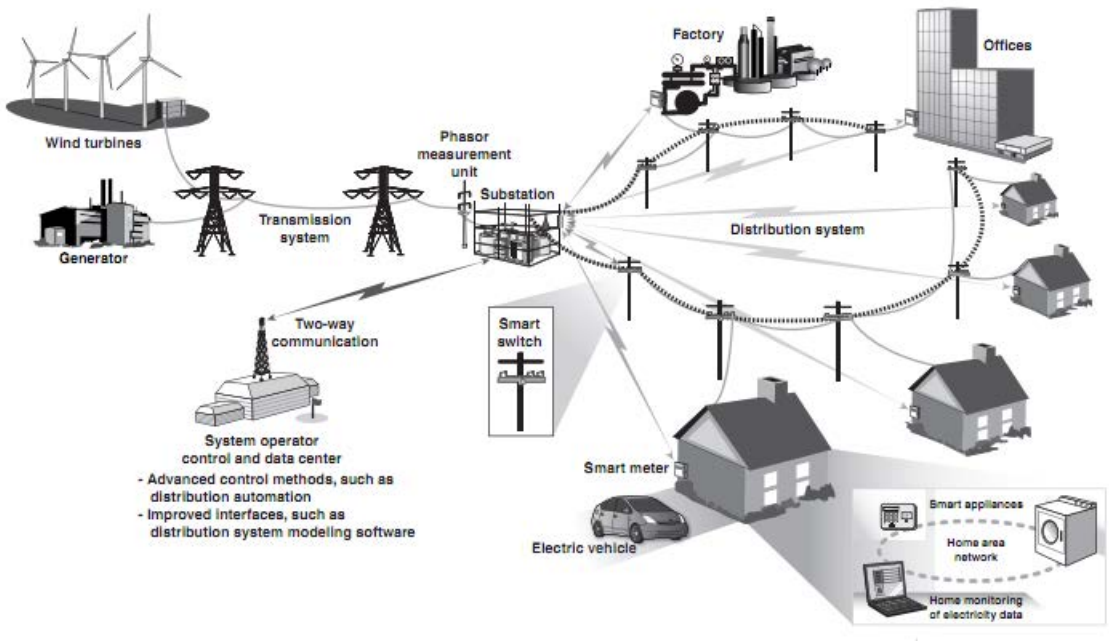

Figure 2.1: Smart grid topography [1].

- Empowering the consumer to a certain extent by allowing him to choose when/how energy is bought;
- Accommodate for both the household consumer and industry needs.

In order to implement activities related to the smart grid monitoring, specific devices are being placed at different geographical locations. The PMU is one such device. It is used as a type of advanced measurement device. A PMU can collect electric metrics, such as the voltage and current phasors at precise instants, through a timing reference given by the Global Positioning System (GPS) [see Chapter 2.2]. This can be tough of as a ”snapshot” of the state in which that portion of the smart grid is in. Using multiple PMUs at different points across the smart grid allows for a complete state estimation. This information can then be used to support decisions on the distribution of current load across the network, which in turn gives us the ability to predict and avoid failures like blackouts. Time synchronization among all PMUs is critical because it allows for the creation of a coherent picture of the smart grid. If ”snapshots” are taken at different times, the complete state cannot be correctly estimated.

Current PMUs use GPS receivers for time synchronization. This is a very costly solution because it requires dedicated hardware to be present in each and every PMU. From a security standpoint, it also leaves the PMUs, and in turn the whole smart grid,

vulnerable to even the most basic GPS attacks like blocking or jamming.

2.2 Global Positioning System

GPS was developed in 1973 by the U.S. Department of Defense, as a result of combining previous navigation systems, some of which were classified. It provides specially coded satellite signals that can be processed in a GPS receiver, enabling it to computer its own position, velocity and the time. Four GPS satellite signals are used to compute positions in three dimensions and the time offset between the receiver clock and the satellite clock. The satellites carry very stable atomic clocks that are synchronized with each other and to ground clocks. Any drift from the true time maintained on the ground is corrected on a daily basis.

The current GPS is composed of three major parts or segments:

- **Space Segment** is composed of all the GPS satellites (also called Space Vehicles). Originally 24 satellites were used, which orbit the earth every 12 hours. They were arranged in such a way that at least six satellites pass over the same location every day. As of December 2012, there are a total of 32 satellites in the GPS constellation. These additional satellites provide redundant information for the receivers, which in turn improve the precision of the GPS receiver calculations;
- **Control Segment** consists of a system of tracking stations located around the world. This system is composed of:
  - a master control station (MCS);
  - an alternate master control station;
  - four dedicated ground antennas;
  - six dedicated monitor stations.

  The MCS is located in Colorado at the Schriever Air Force Base. It measures signals from the satellites and then computer precise orbital data (called ephemeris) and the satellite clock correction for each one. It then uploads the ephemeris and clock data to the satellites;
- **User Segment** consists of all the GPS receivers both civil and military. The navigation in three dimensions is the primary function of GPS. It also allows for extremely accurate time dissemination.

Every GPS satellite continually broadcasts a signal that has two important components:

- A sequence of ones and zeroes called the pseudorandom code that is known by all GPS receivers. The receiver also has the ability of generating the exact same pseudorandom code locally. By comparing the two codes on the time scale (local and received codes) it can calculate the time of arrival of the original code.
- A message that includes the time of transmission of the code epoch (a defined part of the pseudorandom code) and the satellite position at that time.

The GPS satellites transmit data on three different carrier frequencies L1, L2 and L5, the latter only being supported on the newer satellites (at least 5 satellites where launched from 2011, which support the L5 carrier). The data encoding is done using unique code division multiple access (CDMA) [11], so receivers can distinguish between each satellite. This system uses two distinct CDMA encoding types:

- a method based on a code called the course/acquisition (C/A), which is accessible to the general public;
- a method based on a precise (P(Y)) code that is only known and used by the U.S. military and other NATO nations. This code basically encrypts the transmitted data. Devices capable of decrypting this signal are only available to the military and the specifics of the security protocol are kept secret.

The GPS receiver needs to lock on to at least four satellites in order to calculate its position (X, Y, Z) and the time (T). Each message that it receives from the satellite is time stamped (time of transmission). It also gets the time of arrival as previously explained, so it can calculate an estimate of the interval taken for the message to travel from the satellite to itself. Multiplying this value by the speed of light, the GPS receiver, can determine the distance from itself to the satellite.

$$Distance = \Delta T * speedOfLight$$

The time interval calculation (Δ T) is not completely precise due to the clocks of the receiver and satellite not being perfectly synchronized (which is actually impossible considering the atomic clocks that the satellites use). This introduces errors in the process of calculating the exact position (trilateration). This process is shown in [Figure 2.2]. The protocol and all messages that are used are explained in more detail in [12] and [13].

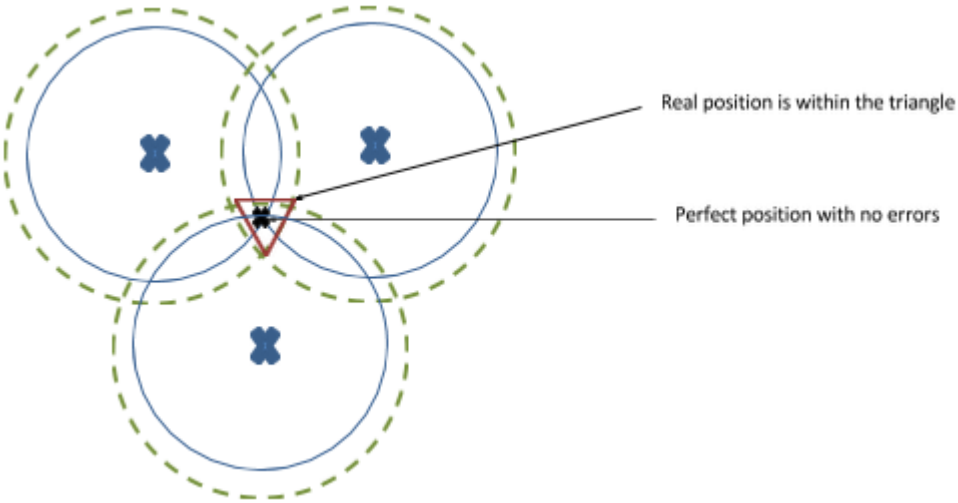

Figure 2.2: Trilateration process used to calculate current position using GPS.

2.2.1 GPS Attacks

The lack of any kind of protection for the civil band (the C/A code) means that many types of attacks exist to try to fool or block the GPS signal. The problem is made even worse by the fact that all the hardware needed for the more advanced types of attacks is readily available to everybody and is relatively cheap considering the circumstances.

The way that the GPS signal source is transmitted also turns it into an easy target for attacks. Because of the huge distance that the signal has to travel, its strength is very low when it reaches the earth's surface. It has often been compared to viewing a normal 20-Watt light bulb from about 19 000 Km away.

The next subsections discuss the most common types of GPS attacks.

GPS Blocking

GPS blocking is the most basic form of attack. This type of attack tries to isolate the receiver from the satellite signal, which in turn leads to a signal loss. It can be performed just by simply covering the GPS antenna of the receiver. It does however require physical access to the GPS receiver in order to be executed which, depending on the adversary, can create some difficulties.

GPS Jamming

GPS jamming attacks [14] [15] are similar to blocking in the sense that they try to make the receiver lose the signal lock. While blocking achieves this by means of isolation, the idea behind a jamming attack is to introduce so much extra radio-frequency noise that the receiver can no longer find the satellite signal out of it. This causes the receiver to be unable to get a lock on the signal.

Considering how weak the satellite signal is, it is not too hard to introduce sufficient radio-frequency noise from a few meters away from the receiver that can completely suppress the original GPS signal. Adding to this, jamming devices are very easy to find on the web and are relatively cheap. Some countries have gone as far as to ban the sale of such devices. It is speculated that the GPS system for an entire city could be prevented from working properly in a flash if the jamming device was sufficiently powerful. To put this into perspective, a device the size of a suitcase could be used from as much as 1000 meters away [16].

GPS Spoofing

GPS spoofing it a more complex type of attack compared to blocking and jamming. It does not block the normal satellite signal but modifies it in such a way to fool the receiver into computing a different erroneous navigation/time solution. The receiver will not be aware of the problem, and therefore it will operate thinking the genuine signal is still being used. This attack can be much harder to detect than jamming.

Some work has already been done in designing spoofing attacks in general [17] [18]. GPS spoofers are also more and more accessible both in terms of cost and of the know-how needed to build them. The manner in which this attack works gives the adversary an extended list of possibilities when it comes to the damage that can be caused. Due to the ”ease of use” and the disastrous results that GPS spoofing can have, some effort is being put into researching, developing and deploying GPS receivers and countermeasures in general, which can cope with these attacks [19] [20].

GPS Software

GPS Software attacks have been recently described [21]. They give the possibility of altering any part of the satellite signal. This can lead to not only the receiver to give an incorrect navigation/time solution, but can even make the receiver crash or downright

become unusable (by exploiting a vulnerability in the executing programs). The fact that manufacturers continue to add features like more connectivity options or even the ability to manage the receiver through the internet, just adds to the problem. GPS as a protocol has limitations from a security standpoint and adding more attack vectors waiting to be exploited does not help to solve the problem.

The attacks demonstrated in [21] show that even high-end GPS receivers can be damaged beyond repair just by modifying a simple parameter in the GPS signal. Again these kinds of attacks are borderline impossible to detect considering the capabilities of the adversary.

2.3 Precision Time Protocol

The precision time protocol appeared as a result of the growing need for high precision time keeping. It was first described in the IEEE 1588-2002 standard, officially entitled "IEEE Standard for a Precision Clock Synchronization Protocol for Network Measurement and Control Systems". A revised version appeared in 2008 as the IEEE 1588-2008 standard [22]. It was designed to fill in the gap left open by the Network Time Protocol (NTP) and GPS. It offers far better accuracy than NTP, but without the need for a dedicated GPS receiver at each network node. GPS receivers can be used in combination with a PTP network by acting as a Grandmaster Clock, i.e., the time source for that network.

A simple PTP network can be seen in [Figure 2.3]. It is composed of a Grandmaster Clock that is the time source for the network; a Boundary Clock that synchronizes itself to this Grandmaster Clock and then further synchronizes other parts of the network; and lastly an Ordinary Clock. Since a PMU uses a GPS receiver to synchronize its local clock, it can be thought of as being made up of a Grandmaster Clock (GPS receiver) directly connected to a Boundary Clock.

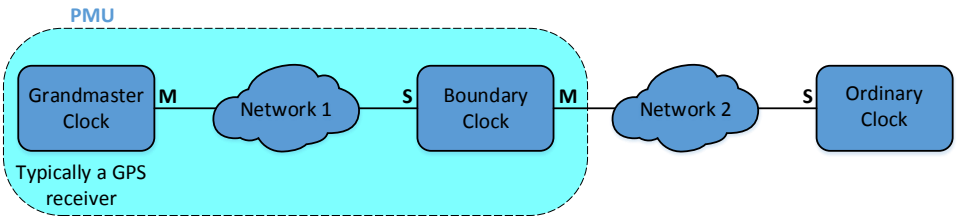

Figure 2.3: Simple PTP Network (M/S means that the communication port acts as master/slave).

2.3.1 PTP Devices

PTP uses the master-slave architecture for time distribution in which one or more communication media are used and one or more clocks. It defines different roles for every clock in the network and various states for every communication port in use:

- **Grandmaster Clock :** is a clock that synchronizes directly to a GPS receiver. It always runs in PTP master mode, meaning it will distribute its time throughout the network;
- **Ordinary Clock :** is a normal clock that synchronizes itself to another source. It always runs in the PTP slave mode;
- **Boundary Clock :** is a clock that both synchronizes itself to a PTP master (acting as a PTP slave) and further distributes the time to another part of the network (acting as a PTP master). **Only the PTP slave port of the Boundary Clock has the ability to change the internal clock**. The other ports can only read the local clock;
- **Transparent Clock :** is a special type of clock that modifies all PTP messages that pass through this device, correcting the message timestamps for time spent traversing the network equipment. This scheme improves the time distribution accuracy by compensating for the time messages spend in each communication device (e.g., a switch).

2.3.2 Message Exchange

The synchronization is achieved by exchanging PTP messages between the master port of a clock and the slave port of another clock. The messages are divided into event messages and general messages. Event messages are timestamped with both transmission and reception times. A message is accurately timestamped in two ways: either on-the-fly as it is about to leave the device (called *one-step mechanism*), or by transmitting another message afterwards containing the timestamp for the first one (*called two-step mechanism*). Only the *Sync message* and the *Pdelay_Resp message* are affected by the used of the two-step mechanism. For the *Sync message*, a *Follow_Up message* is transmitted containing it's egress time. For the *Pdelay_Resp message*, a *Pdelay_Resp_Follow_Up message* is transmitted containing it's egress time. General messages do not require accurate timestamps and are used for both synchronization and configuration purposes. All messages are shown in [Table 2.1].

Based on timestamps, the slave calculates the time offset from the master clock and eventually adjusts its local time to be similar to the master. The basic synchronization message exchange is showed in [Figure 2.4] and the logic behind it is explained below:

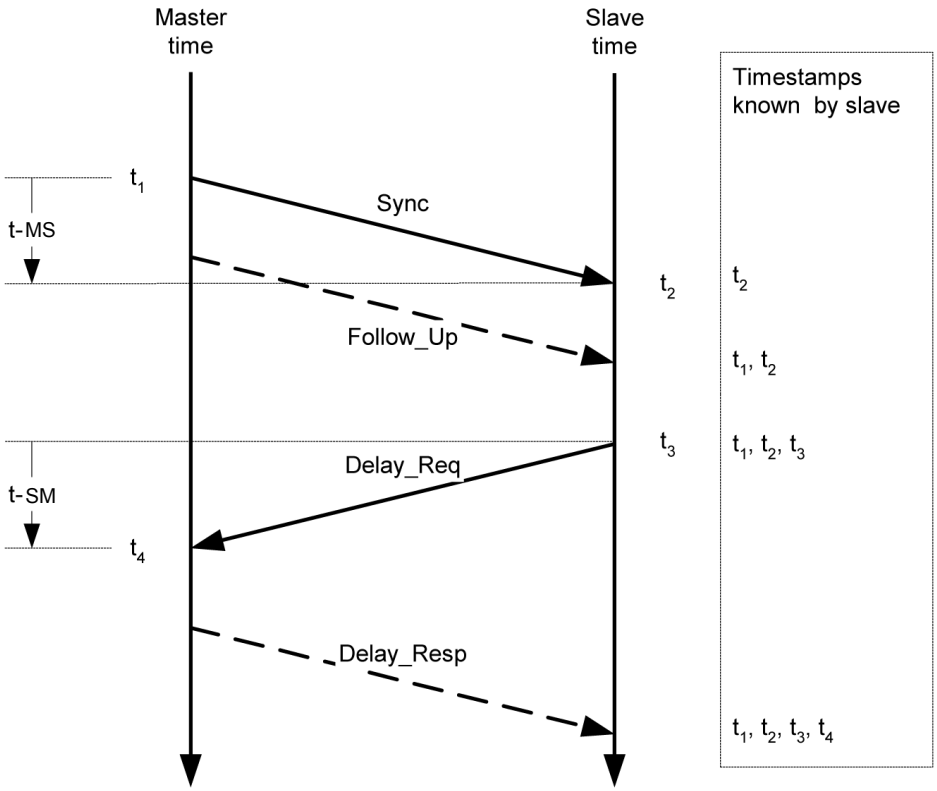

Figure 2.4: PTP Message Exchange.

1. The master sends a *Sync Message* to the slave and saves the time t1 at which it was transmitted;
2. The slave receives the *Sync Message* and notes the time of reception t2;
3. The master conveys to the slave the timestamp t1 by embedding it in a *Follow_Up Message*. Alternatively this timestamp can be embedded into the *Sync Message* using some kind of hardware processing (this is called the one-step mechanism);
4. The slave sends a *Delay_Req Message* to the master and saves the time t3 at which it was transmitted;
5. The master receives the *Delay_Req Message* and notes the time of reception t4;
6. The master conveys to the slave the timestamp t4 by embedding it in a *Delay_Resp Message*.

Type	Message	Description
Event	Sync	Sent by the master. Contains the egress time of the master node
	Delay_Req	Sent by the slave as a response to the Sync message. Contains the egress time of the slave node
	Pdelay_Req	Used to measure the link delay between two clock ports implementing the peer delay mechanism
	Pdelay_Resp	Used to measure the link delay between two clock ports implementing the peer delay mechanism
General	Announce	Contains the clock characteristics like clock quality
	Follow_Up	Delivers the master egress time of the Sync message. Only used in the two-step mechanism
	Delay_Resp	Sent by the master node. It contains ingress time of the Delay_Req message
	Pdelay_Resp_Follow_Up	Used to measure the link delay between two clock ports implementing the peer delay mechanism
	Management	Used for configuration purposes
	Signaling	Used for configuration purposes

Table 2.1: PTP Message types.

2.3.3 Network Delay and Clock Offset

Considering that the clock offset of the slave relative to the master is Δ and that the network delay between the master and slave is *d(MS)* and between the slave and master is *d(SM)* we have:

$$t2 = t1 + \Delta + d(MS) \tag{2.1}$$

$$t4 = t3 - \Delta + d(SM) \tag{2.2}$$

Notice that we have two equations and three unknowns, which prevents us from solving this system of equations. Packets transmitted throughout the network generally suffer different delays in the two directions, and typically there is no way of accurately

measuring the one way delay, i.e., *d(MS)* or *d(SM)*. However, in order to solve these equations an important assumption needs to be made: **the network packet delay is the same in both directions** i.e., *d(MS)=d(SM)=d*. This is an approximation that introduces an error in the calculations, as it is unlikely to hold in a real network. PTP tries to minimize its effect in different ways, such as by taking very accurate timestamps, and by correcting the timestamps when messages go through Transparent Clocks (see Section 2.3.1). Now we can simplify (2.1) and (2.2):

$$d = \frac{t2 - t1 + t4 - t3}{2}$$

$$\Delta = t2 - t1 - d$$

It is now possible to calculate both the one way delay as well as the clock offset Δ. However, because of the aforementioned assumption, the calculated clock offset will have an error that is equal to the difference between the mean delay and the master to slave delay.

There are three main reasons that can lead to asymmetric path delays:

- Operating system latency. This is represented by time the PTP packet spends in the protocol stack. Between the generation of the packet and its transmission on the wire, the packet is manipulated and buffered by the protocol stack. When it reaches the other side, it is delayed by another interrupt. To avoid this problem, PTP packets should be timestamped as close to the physical layer as possible, e.g., at the MAC layer. This approach can reduce the latency introduced by the operating system from milliseconds to nanoseconds.

- Asymmetric link speed. More often than not different links are used for bidirectional communication. Two different links almost never have the same exact speed. This can cause some packets to be received sooner than others, which in turn leads to an error in the final calculation of the clock offset. In order to avoid this, the same link has to be used for both directions.

- Network devices. All network devices (e.g., switches and routers) need to do some kind of processing on the packets they receive. No matter what technique is used, buffering and queuing packets takes time, which increases the delay. To try to

minimize the impact that network devices have on network delay, various methods were devised. The simplest method is to give more priority to the PTP packets. The majority of switches and routers nowadays support packet prioritization. This will alleviate the delay introduced by packet processing. In order to completely eliminate this processing delay, a Transparent Clock can be used. The Transparent Clock compensates for the processing time of the packet by changing the included timestamp.

Another method to try to reduce the impact of asymmetric path delays is PTP delay equalization [23] [24]. It appoints a fixed delay to all PTP packets. All network devices would process the PTP packet as to make sure that all of them experience the same known delay.

2.3.4 Best Master Clock Algorithm

In a distributed environment, an algorithm called *Best Master Clock algorithm* is used to select the best candidate node to serve as the time source. This node will provide the reference time for the network. Selecting the best master takes into account various parameters of each node:

- **Clock quality** - is the expected deviation from real time. In version 2 of PTP, the clock quality is defined with two data fields, the *clockAccuracy* and *clockClass*. The *clockClass* is used for transitioning a port from one state into another in our implementation. By default the values 13, 255 and 248 are used to designate a master only port, a slave only port or a master/slave port respectively. The master/slave port can run in either of these states. In a network of such clocks only one will be master and the rest will be slaves. The BMC algorithm is used to decide which one should be the master. If all clocks are run in master only mode, after the best one is selected, all others will go into a passive state;

- **Priority** - is represented by two 8-bit fields known as *Priority1* and *Priority2*;

- **Variance** - is an estimation of the stability of the clock based on performance observations;

- **Identifier** - is a universally unique numeric identifier of the clock and is used as a tie breaker if all the other parameters are the same.

Each clock uses an internal data set to save its own parameters and *Announce messages* are used to send this information from the current best clock (current master) to all other connected clocks. Whenever a *Announce message* arrives at a node, a comparison is made between the internal parameters of the receiving clock and the data set included in the *Announce message*. After this, a decision is made best on the data set comparison. As such the Best Master Clock algorithm can be divided into two phases:

- Data set comparison phase. In this phase the proprieties of two different clocks are compared, as indicated by their respective data sets. The input is represented by data set contained in the received *Announce message* and the internal data set of the clock that received that *Announce message*. The output is which one of the two clocks is the most suited to take the role of master.
- State decision phase. After the data set comparison ends, a decision needs to be made as to what is the most recommended state of the internal clock. If the other clock from the comparison is better, the internal clock becomes a slave (or goes into a passive state). Otherwise it becomes the master.

2.3.5 PTP security

Annex K of the Precision Time Protocol standard IEEE 1588-2008 [22] presents several security guidelines that can help the protocol withstand different types of attacks. The guidelines are however completely optional and in an experimental state. They focus on adding two main security mechanisms:

- An integrity protection mechanism, which uses a Message Authentication Code (MAC) to verify that messages have not suffered any unauthorized modification in transit. A message counter is also implemented to help prevent replay attacks;
- An authentication method based on a challenge-response mechanism.

A flag in the PTP message is used to indicate that it is carrying security related information. Each message following this one is required to also present the same flag or else it is silently discarded without wasting any more resources. In order to ease this process for hardware implementations, the flag field should be the last field in the message. By doing this, an indirect protection against Denial of Service attacks on system resources is created. In [25] a comprehensive threat analysis of the PTP protocol is done and solutions using IPsec and MACsec are explored.

Chapter 3

Fault Tolerant PTP

At the time of this writing, PTP has already been researched as a solution for time synchronization in the smart grid environment. However, the main concern of these studies is the clock synchronization accuracy that can be achieved. The security problems of this protocol have yet to be addressed. In this chapter we present a new approach to employ PTP in a smart grid environment. We also discuss the changes to the PTP standard that address the lack of security, when the primary attack vectors are the GPS receivers.

3.1 Overview

Some of the challenges of deploying PTP in a smart grid network are similar to those presented in [26]. Our solution is based on a PTP Boundary Clock [Figure 3.1]. It synchronizes directly to a Grandmaster Clock and then distributes this time information to a part of the network.

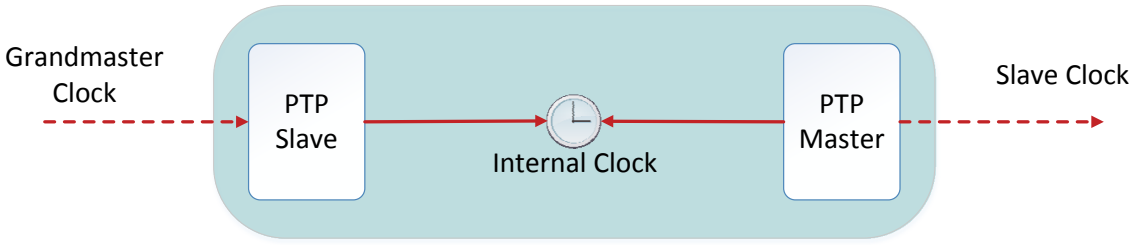

Figure 3.1: Boundary Clock as originally defined by PTP.

Our modified Boundary Clock [Figure 3.2] acts like a normal Boundary Clock but has an additional port that functions as a backup in the case the Grandmaster Clock fails.

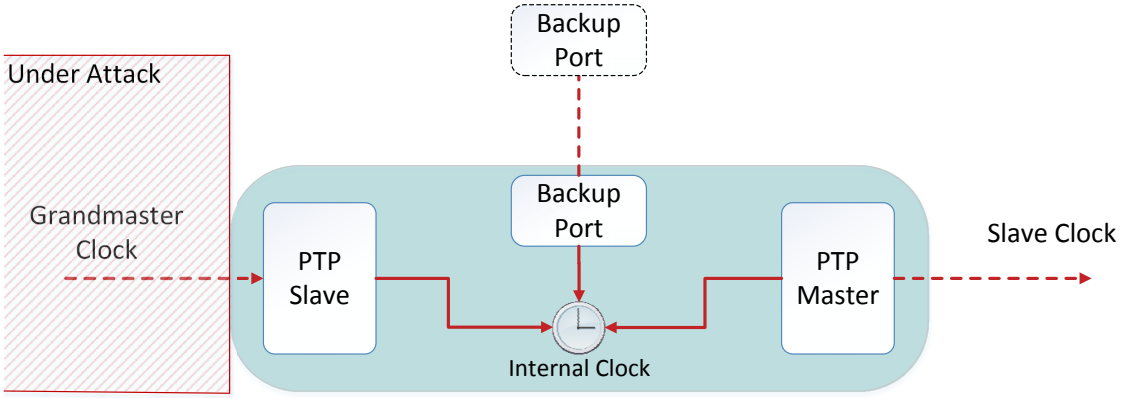

Figure 3.2: Boundary Clock with the proposed modification.

The backup port takes the role of the PTP master once the node identifies misbehavior of the Grandmaster Clock, ensuring that the internal clock remains synchronized with real time. This backup port will be connected with other such ports from various Boundary Clocks around the network. To ensure full time synchronization among all nodes, we need to have several GPS based Grandmaster Clocks and Boundary Clocks spread throughout the network to ensure some geographical distribution. However, if the network delays and jitter allow it, we can use the same GPS receiver for more than one node. Not only does our solution potentially manage to reduce the number of GPS receivers needed but it also supports the detection and mitigation of GPS attacks. This enables the node to always synchronize itself to a time source even if it is not as good (or precise) as the original.

3.2 Attack Model

Our focus is on detecting and mitigating attacks on the GPS receiver (i.e., Grandmaster Clock) that offers time information to the boundary clock as well as on the communication link between them. There is an initial prerequisite that **all boundary clocks are synchronized to a correct time source before any attacks occur**.

Fault tolerance represents one of the most important concepts in distributed systems design. It enables a computer system to continue operating normally even when one or more of its components fails. This does not imply that the failures do not have any consequence on the system. The computational cost of maintaining the system running

correctly in this case is directly proportional to the number of failures the system is suffering. However, in systems with a high degree of fault-tolerance the impact of failures on performance can be eliminated entirely.

We are interested in two main types of failures: crash and arbitrary. Crash failures are self-explanatory. When a system crashes it is unable to continue providing the service it was initially intended to offer. Arbitrary failures encompass all sort of erroneous behaviors that the component might experience, including delays in producing responses or the transmission of invalid results. Arbitrary failures are also called Byzantine failures referencing the Byzantine Generals Problem [27].

We abstracted the ways in which the Grandmaster Clock can fail into two classes, as perceived by the Boundary Clock:

- **Crash failures** are observed when the Grandmaster Clock is incapable of sending any type of information to the Boundary Clock. This can happen when the GPS receiver of the Grandmaster Clock is:
  - under a Blocking or Jamming attack and loses its satellite signal lock;
  - under a Software Attack that crashes the GPS receiver;
- **Byzantine failures** are assumed when the information received from the Grandmaster Clock is incorrect in any way, or when it reaches the Boundary Clock late or in an arbitrary order. This can happen when the GPS receiver of the Grandmaster Clock is:
  - under a Spoofing Attack that causes the Grandmaster Clock to give usable but incorrect data;
  - under a Software Attack that renders the Grandmaster Clock unable to provide any kind of usable information.

  A Byzantine fault is also assumed when the Communication Link interconnecting the Grandmaster Clock to the Boundary Clock is under attack, which in turn causes the time information to be delayed or to reach the Boundary Clock in an incorrect order.

3.3 System Architecture

We define three new port states based on the existing three states of PTP Master, PTP Passive and a PTP Slave. The new port states are called Fault-Tolerant (FT) PTP Master, FT PTP Passive and FT PTP Slave respectively [see Figure 3.3]. As previously mentioned, our modified Boundary Clock distinguishes itself from a normal one by the presence of one extra port that will interconnect various such Boundary Clocks.

- **FT PTP Slave** has the purpose to detect abnormal activity coming from the Master port it is connected to (in our case this will be the Grandmaster Clock port) and to alert the backup port. It does this by analyzing the time information coming from the Grandmaster Clock, as well as the network delay between them;

- **FT PTP Master** and **FT PTP Passive** are port states in which the newly added backup port of the Boundary Clock will run. Their primary purpose is to distribute time information (if the Grandmaster Clock where it synchronizes itself is still considered correct) to other Boundary Clocks.

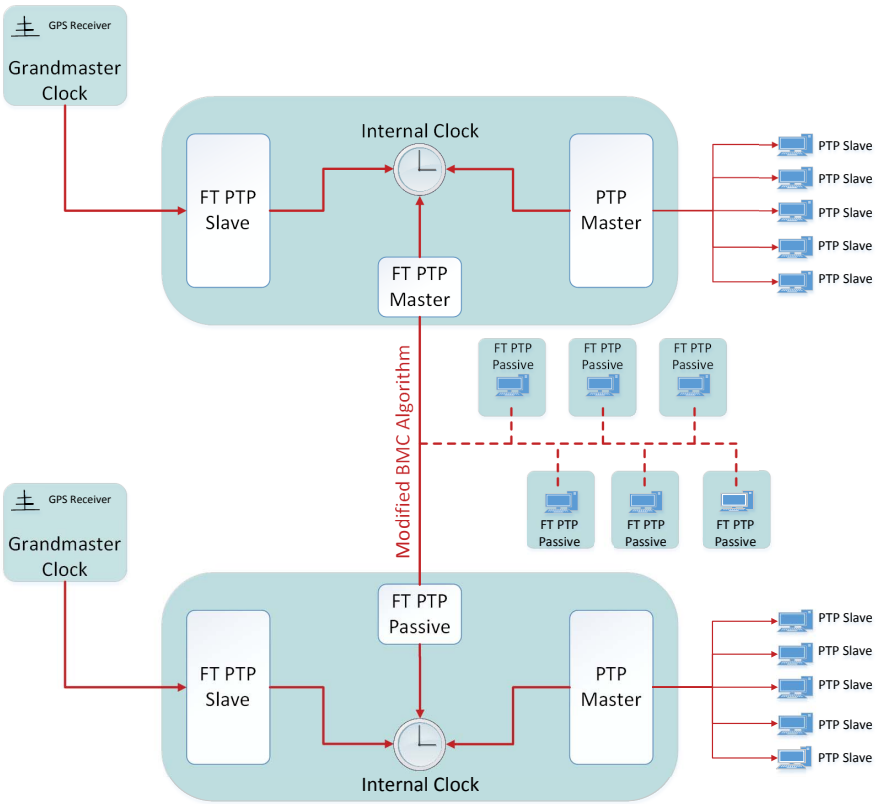

Figure 3.3: System Architecture.

The normal PTP Slave port of the Boundary Clock is replaced with a FT PTP Slave. As mentioned before, the backup port of the Boundary Clock will either run in FT PTP Master or FT PTP Passive modes. A modified version of the Best Master Clock algorithm runs between the backup ports with the responsibility of deciding which one is the FT PTP Master. After the best FT PTP Master is selected, all the other ports will go into FT PTP Passive mode. The modified version of the BMC algorithm takes into account another parameter used for disqualifying one such port from the algorithm (see below).

3.4 Solution

In this section we describe how the detection and recovery mechanism works. The state of the Boundary Clock during normal operation and when under attack are displayed in [Figure 3.4] and [Figure 3.5].

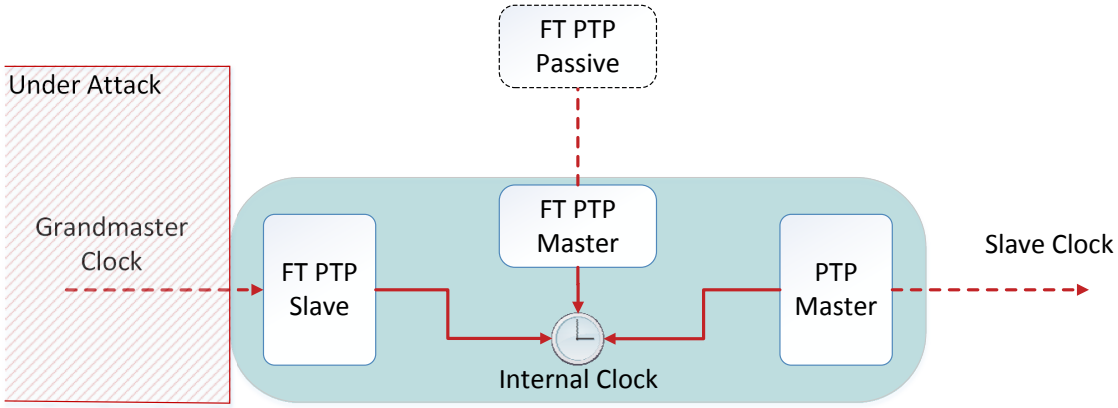

Figure 3.4: Boundary Clock state during normal operation.

When an attack is detected by the FT PTP Slave, it first does a clock jump[1] of *2 * DEFAULT_SYNC_INTERVAL* (1 second in our implementation) ahead, then proceeds to disable itself. The backup port (running in either FT PTP Master or FT PTP Passive) observes the clock jump by reading the clock at each full run of the protocol. Next, it goes into a normal PTP Slave mode. Any out of the ordinary modifications to the internal clock of the Boundary Clock are detected and handled in this way.

The backup port can be in two different states when an attack is discovered: FT PTP Master or FT PTP Passive. As mentioned, in both cases the backup port will go into PTP

[1] A sudden change in the order of milliseconds or larger of the internal clock time.

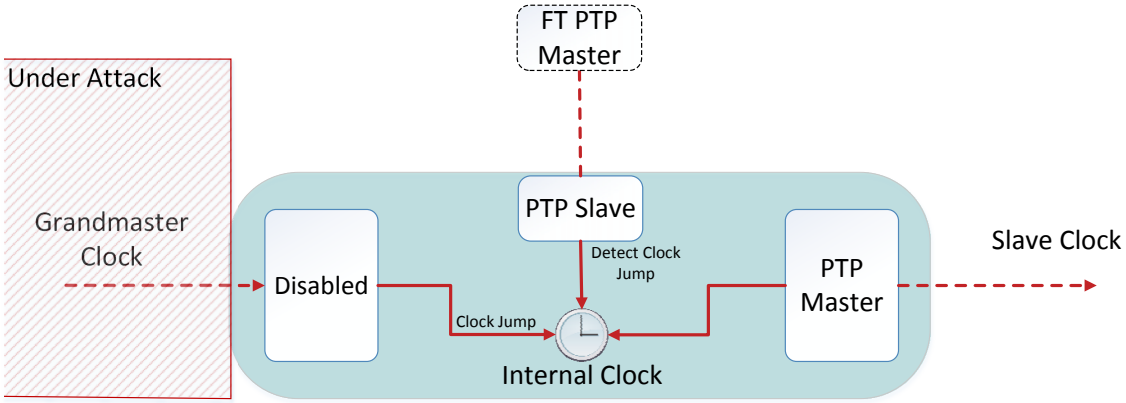

Figure 3.5: Boundary Clock state after an attack was detected.

Slave mode. The difference consists in the fact that if the current state is FT PTP Master, another backup port will need to be elected to go into FT PTP Master state. A modified BMC algorithm is used to perform this change, by comparing data from all the backup ports of the Boundary Clocks.

Notice that we use the local clock of the Boundary Clock as a means to communicate failure detection between the two ports. This is done in order to minimize changes to the standard. In alternative, a low level, high speed communication scheme as presented in [28] could help with the recovery time.

The clock jump performed by the FT PTP Slave will always have to be greater than the *DEFAULT_SYNC_INTERVAL*. This is done to make sure that the backup port (now in PTP Slave state) will receive at least one *Sync Message* from the new time source (in this case the backup port who was elected as the new FT PTP Master from another boundary clock) to compare the origin timestamp from this *Sync Message* to the current local clock time.

The state change from FT PTP Master to PTP Slave is carried out by modifying the *clock_class* value of this port from 13 (associated with a Master only port) to 255 (Slave only port). Once into PTP Slave mode this port will start synchronizing to the FT PTP Master and the previously set local clock jump will be immediately detected and corrected. In the worst case scenario the recovery time is *DEFAULT_SYNC_INTERVAL* + *PTP Slave Initialization Time*.

Next we describe in more detail how the detection mechanism implemented in the FT PTP Slave works for crash and Byzantine faults.

3.4.1 Crash failure detection

The crash failure detection mechanism is based on the periodic transmission of *Announce Messages*. The normal behavior of the Grandmaster Clock is to send an *Announce Message* every *DEFAULT_ANNOUNCE_INTERVAL* (1 second in our implementation). The FT PTP Slave keeps track of the received announce messages and waits a maximum of *announceTimeoutGracePeriod* (with a value of 3 seconds) before declaring a packet as lost. We consider the Grandmaster clock failed if two *Announce Messages* are lost. This translates to a maximum wait time of 6 seconds, which is a lot but serves as a means of testing and verifying the algorithm. Of course, the bounds could be made tighter but with the cost of an increased performance overhead.

3.4.2 Byzantine failure detection

As a first detection layer, all information reaching the Boundary Clock from the Grandmaster Clock is filtered to eliminate any bogus data. This will prevent most arbitrary behaviors that cause the transmission of unexpected data. From the previously mentioned attacks, the only one that can do this is the GPS Software attack. This scenario encompasses most of the attack vectors but two cases need to be treated separately because of their volatile nature:

- **Attacks on time.** In this type of attack the goal is to try to provide inaccurate time information to the Boundary Clock to desynchronize it. The primary information used to detect these attacks comes from *Sync Messages*. An attack can occur in two different ways: either the provided time value jumps forward by a significant amount in one step (clock jump), or the clock is skewed[2] gradually step by step. In order to detect both types, we save the values of the clock offset between the Grandmaster Clock and the Boundary Clock after the clocks are synchronized (Original offset), and the offset calculated in the last execution cycle of the algorithm, which is based on the latest received *Sync Message*. We know that the values will vary very little considering that clocks are already synchronized (prerequisite). So, if an attacker tries to jump the clock this will be discovered by comparing the original offset with every new calculated offset. The values used to define the maximum allowed offsets are: *MAX_SECONDS_ALLOWED_OFFSET*, and *MAX_ALLOWED_DRIFT (in ns)*. The first one is used to detect offset jumps

[2] A change in the order of μs or slower relative to the internal clock time.

in the order of seconds. The second one is employed to either find clock jumps by comparing the last known offset with the newly calculated one, or to discover skews by comparing the original offset with the newly calculated one. Special care needs to be given to the *MAX_ALLOWED_DRIFT* parameter because it represents the maximum value that the Boundary Clock time is allowed to change at each execution cycle. If an attack increases the clock offset at a rate slower than the *MAX_ALLOWED_DRIFT* parameter, time data from other backup ports can be used to detect it. Two maximum offset values were used because time is represented by two fields (seconds and subseconds field) in our implementation. The *MAX_SECONDS_ALLOWED_OFFSET* and *MAX_ALLOWED_DRIFT* parameters need to be adjusted by measuring and averaging the clock offset and network delays between the Grandmaster Clock and Boundary Clock.

In order to detect clock skews smaller than the *MAX_ALLOWED_DRIFT* value we could make use of the communication link between the different backup ports to exchange time information. In this scenario the backup port would need to be in either FT PTP Master or a newly defined state. The objective of this new state would be to measure the clock offset of the internal clock compared to other Boundary Clocks. It could even make use of a secondary internal clock of the Boundary Clock that would be inaccessible by any other means. Only this new state of the backup port would have the ability to modify this secondary clock. This is a Master-Slave scheme similar to the normal PTP where the new state would be responsible for disciplining (modifying the clock in such a way as to synchronize it's time with another clock) the second clock. This new clock would give us the ability to detect even the smallest of skews (even the drift values) to the first internal clock. The problem is that there is no way of detecting if the Master (in this case, the backup port running in FT PTP Master from another Boundary Clock) is under attack. This means that time attacks consisting of skewing the clock slower than the defined *MAX_ALLOWED_DRIFT* would only work on the Boundary Clock whose backup port runs in FT PTP Master. This is still a huge improvement over having no protection at all. Further work can be done to test and evaluate the validity of this approach because it depends highly on the network conditions of communication link between the Boundary Clocks.

- **Attacks on the communication link.** As previously mentioned the attacker has total control over the communication link between the Grandmaster Clock and the Boundary Clock. This allows him to delay or drop PTP packets. Dropped packets

will make the Boundary Clock think the Grandmaster Clock has crashed, causing the usual recovery actions to be performed.

Delay detection is done in a similar way to the detection of attacks on time but using the *network delay* as primary information. The delay is calculated by means of the standard delay request-response mechanism [Section 2.3]. Once again we save the first calculated delay value after the clocks are synchronized and the one computed in the last cycle of the algorithm. The main difference consists in the fact that network delays can vary more than the clock offset. Therefore, instead of immediately considering the Grandmaster Clock failed, we consider deviations from the normal delay to be outliers. For a delay value to be considered an outlier it has to be greater than *MAX_OUTLIER_DEVIATION (in ns)*. When a sufficient number of outliers (*MAX_DELAY_OUTLIERS*) are detected over a period of time, the Boundary Clock will consider the Grandmaster Clock failed and act accordingly.

3.4.3 Modified Best Master Clock Algorithm

As previously mentioned the Best Master Clock algorithm is used to select the clock that is most suited to distribute time information throughout the network. *Announce messages* carry information that describes a clock state and quality. As such all clocks use an internal data set to save their own attributes and compare it to the received data set from the *Announce message*.

Our modification consists in the addition another parameter called *isCandidate* that is verified at each run of the algorithm. This parameters is used to indicate that a clock is no longer suitable for being a master. This happens when a backup port of a Boundary Clock has detected an attack and goes into slave mode. In addition to the internal data set of the clock, the *isCandidate* parameters of all the participating clocks also need to be saved internally.

Announce messages only matter when they carry information about the current best clock. The *isCandidate* parameter however has to be verified independently of the state in which a clock is. This happens because even a clock in FT PTP Passive state can fail and needs to be disqualified. Under normal circumstances this would not happen.

Chapter 4

Implementation and Experimental Evaluation

To evaluate the feasibility of our solution, a prototype was implemented based on a software-only open-source implementation of PTP. The main goal was to evaluate and measure the performance impact of the changes we have made to the PTP standard. The results of our experimental tests are presented in last part of this chapter, where they demonstrate an added capability for the detection of failures.

4.1 Implementation

The open source ptpd2 [10] was used as a base for our implementation. It was chosen because it is specifically tailored to be used as a software only implementation [29] without any hardware assistance. In [Figure 4.1] we present a UML diagram explaining the structure of the software. Note that the diagram does not represent all the components of ptpd2 (e.g., it excludes the NTP failover component of ptpd2 that kicks in when no time sources are available for synchronization) but rather those that have been of interest to us.

The first thing that happens when the ptpd2 software starts up is the creation of the two main data structures of the protocol: a *RunTimeOpts* structure and a *PtpClock* structure. The first one is used to save the immutable initialization parameters of the protocol. This data structure is passed as argument to the classes whenever something needs to be initialized or verified. The second data structure, the *PtpClock*, is a representation of a PTP port. It is used to save information about the current running instance of ptpd2.

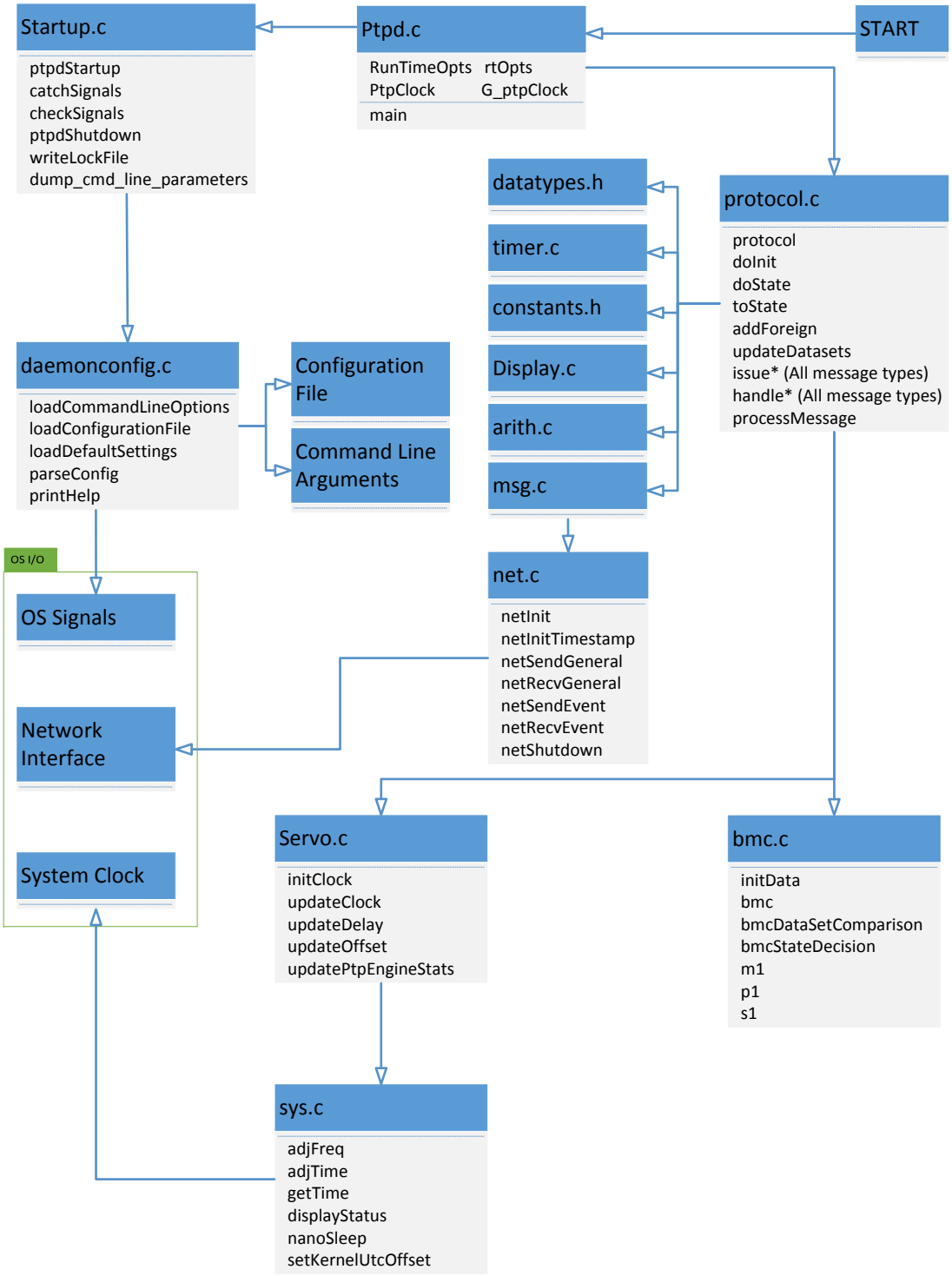

Figure 4.1: Ptpd2 class diagram.

After this step, the configuration file and command line arguments are read by the *daemonconfig* class. It is at this point that alot of verifications are made about the validity of the input parameters. If everything is fine the *daemonconfig* class passes this information to the *Startup* class which will initialize the *RunTimeOpts* data structure and allocate

the necessary memory for the *PtpClock* data structure (*ptpdStartup* method). Some parameters regarding the operating system kernel and platform are also initialized in the *PtpClock* data structure at this point.

Before the protocol starts, a lock is put on the currently running instance of the software, by creating a lock file, as to block concurrency issues that may arise with having multiple instances of ptpd2 changing the internal clock at the same time. Ptpd2 does not yet support Boundary Clocks per se, because of the way that the *PtpClock* data structure was constructed. However, we managed to workaround this by running two instances of the software with different locks for each instance. This allowed for the creation of a Boundary Clock. Great care needs to be taken with this approach to avoid concurrency problems. In our case, only one of the ports will be allowed to change the value of the internal clock at a given time, but both can read it simultaneously.

Next, the main loop of the protocol is started (the *protocol* method of the *protocol.c* class). The first thing that happens is the data initialization of the main *PtpClock* structure with parameters from the *constants.h* file as well as the runtime parameters from the RunTimeOpts data structure. This is done only once, when the protocol starts, by the *doInit* method.

Next we describe the roles of each of the classes included in the class diagram:

- *datatypes.h*: contains the definition of all the data structures used by the protocol;
- *timer.c*: defines the concept of timer for the protocol. Timers are used when waiting for network PTP messages to arrive. This allows for the detection of dropped packets;
- *constants.h*: defines the default initialization values for various data structures used by the protocol;
- *Display.c*: implements the general rules of representing data both to the standard output console and to log files;
- *arith.c*: consists of time format conversion routines and additional math functions needed by the protocol;
- *msg.c*: is used for message packing and unpacking for network transmission. It contains individual packing and unpacking methods for each defined message type;

- *net.c*: is the class used for network communication. It is responsible for network discovery, packet timestamping and netowrk packet transfer. When packets reach this phase they are ready/packed for network transmission. This class represents the lowest level of message timestamping available in software implementations of the PTP standard. In order to improve accuracy, packets can be timestamp at the physical layer by the *Network Interface* itself in hardware assisted or hardware only implementation of the PTP standard;

- *bmc.c*: implements the BMC algorithm. It divides the implementation into two parts as explained in Section 2.3.4: the data set comparison phase (implemented in the *bmcDataSetComparison* method) and the state decision phase (implemented in the *bmcStateDecision* method). The *initData* method runs only when the protocol starts for the first time and is responsible for initializing all the *PtpClock* run-time parameters that are used in the BMC comparison (see Section 2.3.4). The last three methods *m1*, *p1* and *s1* are used to copy the data sets (received in an *Announce* message) into the local *PtpClock* when a state change occurs. *m1*, *p1* and *s1* are used when the local *PtpClock* goes into master, passive or slave port respectively.

- *Servo.c*: is the class responsible for disciplining the internal clock and calculating the delay and offset. The whole process can be seen in 4.2 and is explained in more detail in [29];

- *sys.c*: contains all the code to call the kernel time routines;

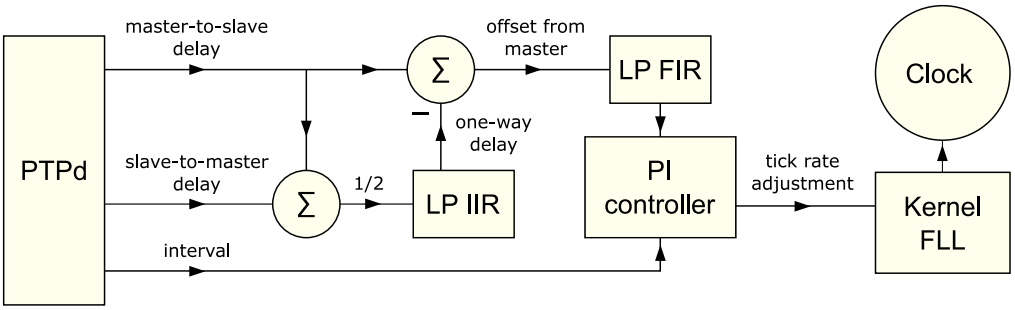

Figure 4.2: Ptpd2 clock servo diagram.

All the classes that we ended up modifying can be seen in 4.3 in red.

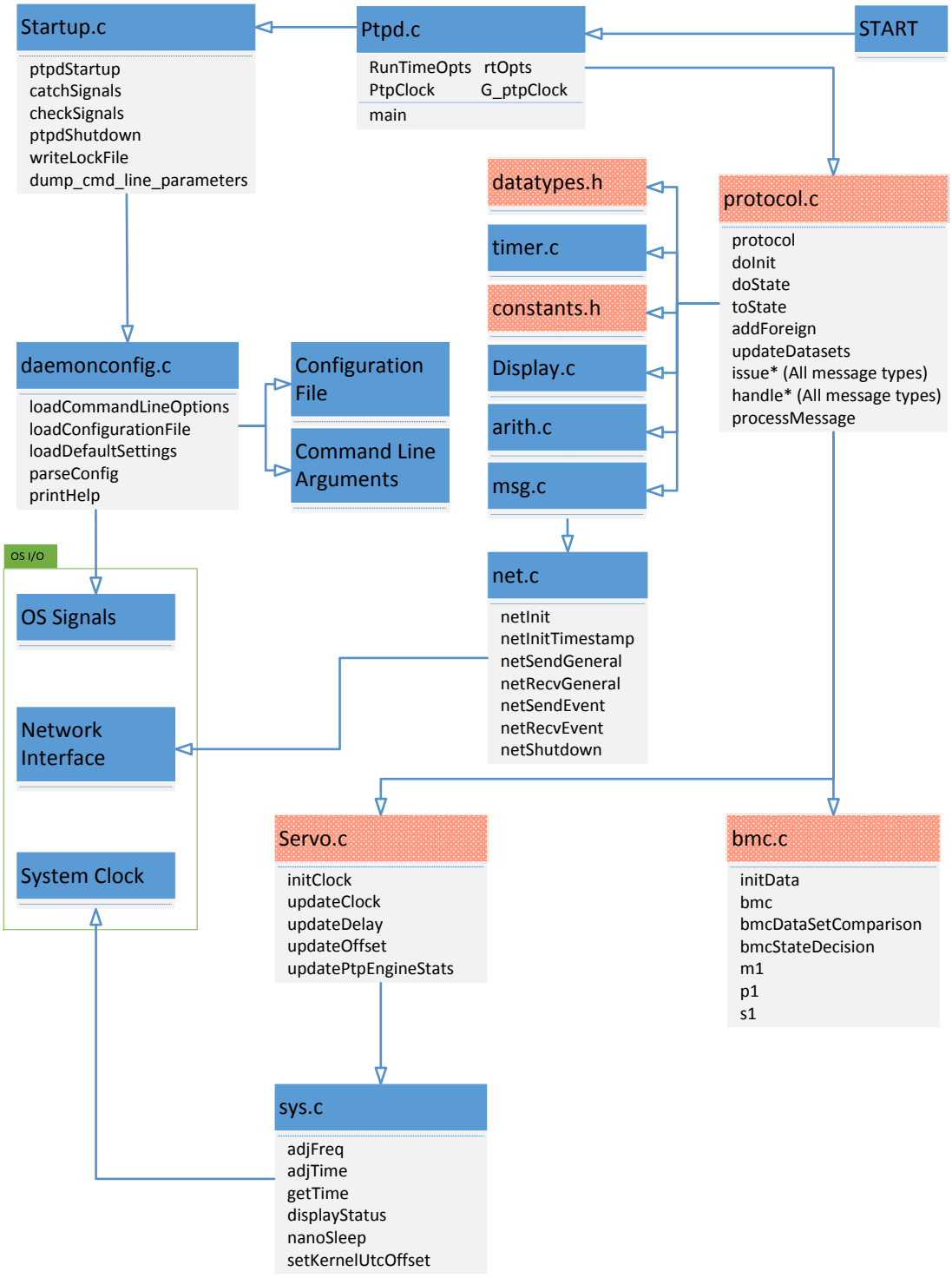

Figure 4.3: Modified ptpd2 class diagram.

Next we explain the most significant modifications that we made to every class:

- *datatypes.h*: three data structures were modified here:
  - *AnnounceMsg*. We added a new parameter *isCandidate*. This parameter is used to disqualify the backup port of a Boundary Clock from the modified BMC algorithm.
  - *ForeignMasterRecord*. This data structure is used to save information about all other known PTP nodes. The *isCandidate* parameter was added as well as a parameter to save the timestamp of the last received Sync message called *lastSyncTimestamp*;
  - *PtpClock*. Was modified to save parameters that are used for evaluation purposes: *isCandidate* (representing the isCandidate value of the local clock); *previousDelay* and *previousTime* that represent the internal clock time and delay calculated in the last execution cycle of the protocol; *originalTimeOffset* utilized for verification purposes, to disallow the clock to drift too far from its master; *delayOutliers* is the number of detected delay outliers up until a certain point (see Section 3.4.2) and lastly *lastSyncTimestamp* is the timestamp from the last received Sync message from the current master;

- *constants.h*: the default values for the attack detection tolerances are defined here. The two variables that are used to activate/deactivate the attack detection mechanism are also defined here: one for attacks that target time information (*FT_TIME*) and one for attacks on the network communication link (*FT_DELAY*);

- *protocol.c*: contains most of our modifications. All the attack detection mechanisms are verified here, at each run of the main loop;

- *bmc.c*: contains modifications that consist mostly of the verification of the new *isCandidate* value. This is where this new parameter is copied to and from the *Announce* message;

- *Servo.c*: allows for a master or passive port (originally they were never allowed to go into slave mode by the configuration) to modify the internal clock. Normally this can only be done by a slave port. However, we only allow a master or passive port to do this if it has transitioned into a slave port. This is what happens to the backup port of the Boundary Clock when an attack is detected.

4.2 Testing Platform

For the testbed three off-the-shelf PC's running Ubuntu 14.10 were used. One of the machines was equipped with two network cards (one for the FT PTP Slave port and one for the FT PTP Master/FT PTP Passive port) and played the role of the Boundary Clock. The other two machines simulated a Grandmaster Clock (running a normal PTP Master) and a second Boundary Clock with a backup FT PTP Master/FT PTP Passive port, to act as a new time source when the first one fails. We did not simulate the other normal PTP Master port of the second Boundary Clock and the network it synchronizes because it is just a normal PTP network.

Since we used off-the-shelf PC machines, the internal oscillators of the clocks are of low quality. If left unsynchronized, they drift apart by a large amount in a relatively short span of time. This would not happen in a real smart grid network because the PMUs oscillators are of a better quality. The test network was a normal 100 Mbps Ethernet LAN with a lot of nodes, traffic and jitter. It is safe to assume that the network conditions in a smart grid environment will be considerably better. However all nodes are relatively close geographically, closer than in some deployment scenarios of a smart grid network.

4.3 Results

We experimented our algorithm in three test cases. Our first test case consisted in injecting a Crash fault [Figure 4.4].

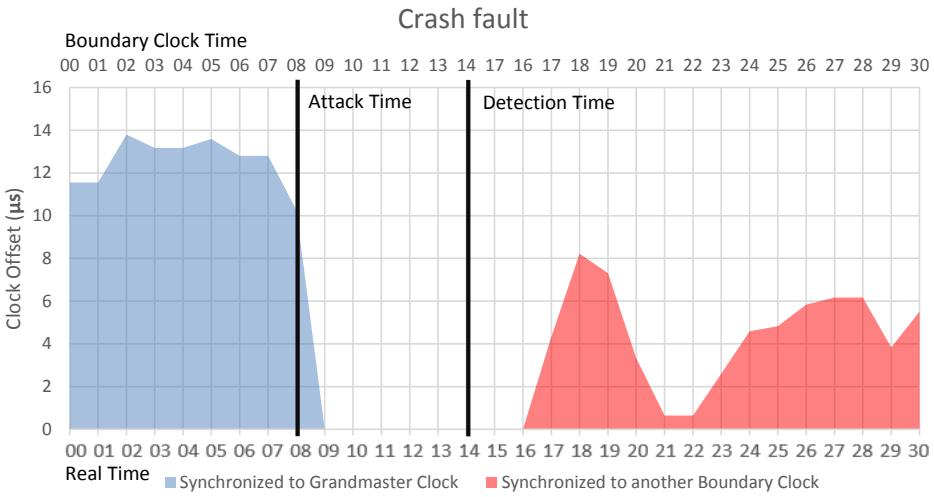

Figure 4.4: Crash fault detection.

This was simulated by simply killing the PTP process running on the Grandmaster Clock machine. This was done at time ”08”. After exactly 6 seconds (as previously described *2 * DEFAULT_ANNOUNCE_INTERVAL*) the Grandmaster Clock was considered failed. At that moment the FT PTP Slave port of the Boundary Clock increased the internal clock by 2 seconds and proceeded to disable itself. The FT PTP Master/FT PTP Passive port now detects the clock skew and goes into PTP Slave mode. After receiving the first *Sync* message from a FT PTP Master port of another Boundary Clock the synchronization process starts.

The second test case is an attack on time. This can happen in two ways: either by skewing the clock or by jumping it a large amount at once. We present results for the first case, which is harder to detect. The second case detection scheme is similar except we compare the current local time to *MAX_SECONDS_ALLOWED_OFFSET* instead of *MAX_ALLOWED_DRIFT*. The results would also be similar because nothing else changes.

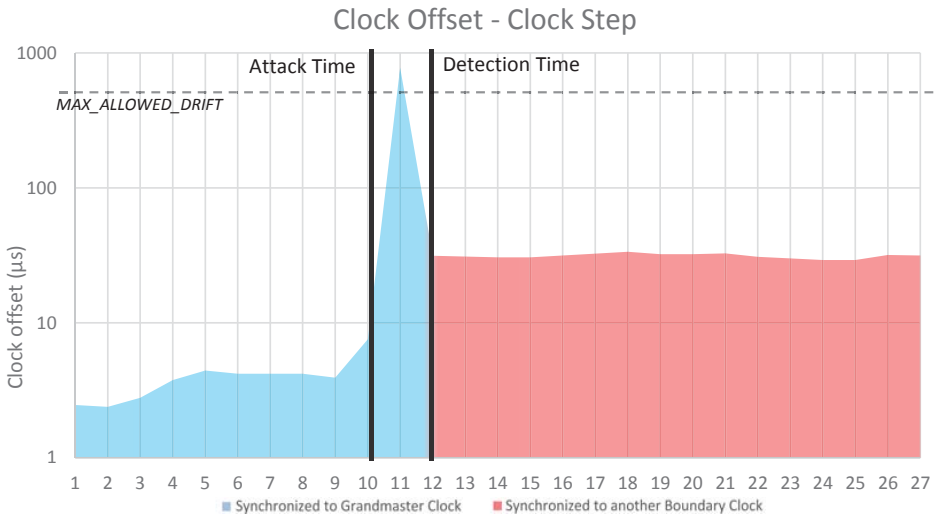

Figure 4.5: Gradual Clock Skew.

A bash script was created and ran on the Grandmaster Clock machine. It will skew the clock by a small amount by performing a small set of simple operations. The script reads the value of the clock, saves this value into a variable then proceeds to set the clock to the value of that variable. This works because it takes $\simeq$ 1 ms to set a variable in memory in our machines. [Figure 4.5] shows the perceived increase in the clock offset of the Master (appearing on a logarithmic scale in the graph) that is suffering the attack (left side of the graph) . The value for the *MAX_ALLOWED_DRIFT* parameter used for this test was 700 μs.

The last case we tested was when the attacker delays PTP packets sent by the Grandmaster Clock. The parameter *MAX_OUTLIER_DEVIATION* had the value of 300 μs and the *MAX_DELAY_OUTLIERS* was set to 3. To simulate the delays, we employed the *tc* command [30]. The *tc* command is used to show or manipulate traffic control settings. This allowed us to introduce a 450 μs delay in the network.

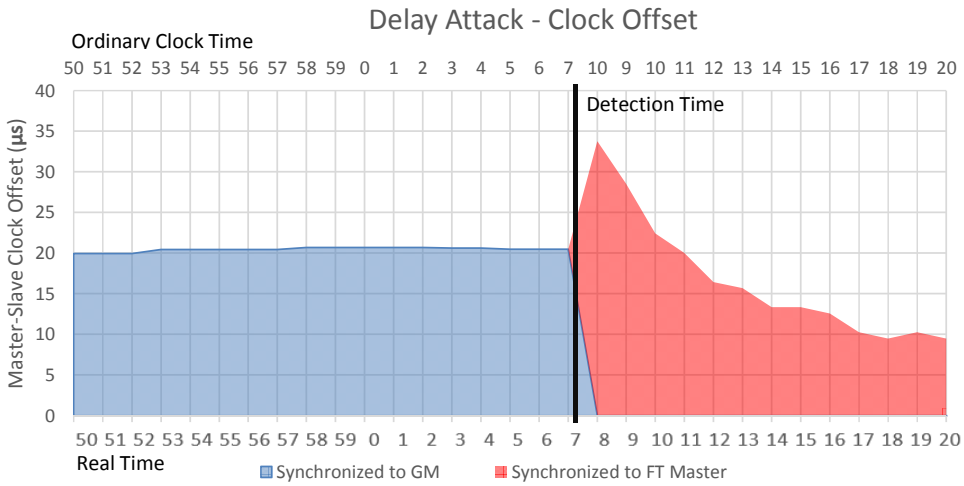

Figure 4.6: Clock offset during Delay Attack.

The clock offset perceived by the Boundary Clock is shown in [Figure 4.6]. We can see that after recovery the clock offset start normalizing with even better accuracy than while synchronizing with our simulated Grandmaster Clock. This happened because all of our machine are experiencing almost the same network conditions. In a real smart grid environment the perceived clock offset after recovery will always be greater than the initial offset.

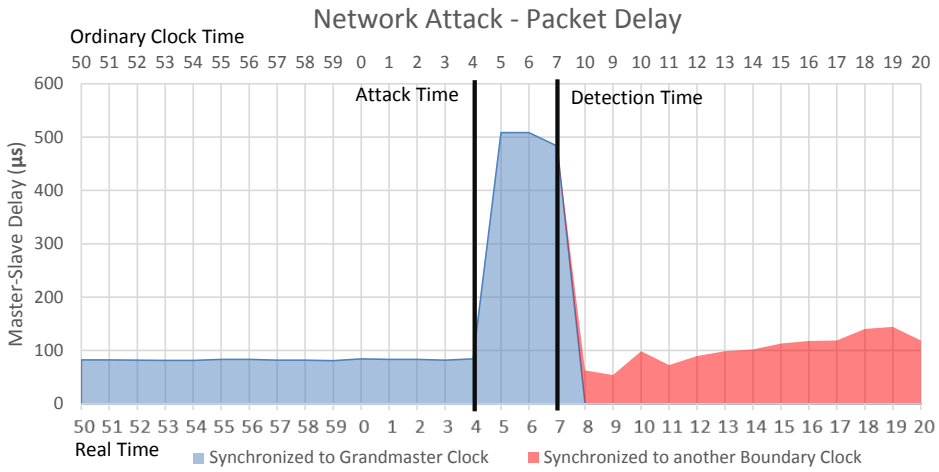

Figure 4.7: Packet Delay Attack.

After recovery, we can see from the graph [Figure 4.7] that the delays of the new

communication link between the FT PTP Master/FT PTP Passive (now in PTP Slave state) port of the Boundary Clock and a FT PTP Master port from another Boundary Clock are not as stable as before the attack (where the Grandmaster Clock corresponding to the GPS receiver was synchronizing the FT PTP Slave port (now disabled)). This is due to the fact that the delays are calculated based on the clock offset and the first Boundary Clock is not yet fully synchronized to the new time source. After some time they will normalize and again become stable.

Chapter 5

Conclusion

The evolving needs of the smart grid bring a new set of problems that has to be addressed. Time synchronization in such an environment is a critical necessity. This requirements is currently realized by the use of various GPS receivers, which creates a large attack surface for attackers to exploit.

In this work, PTP was used as the primary building block for a novel solution. An approach based on PTP was developed, taking the necessities of the power grid environment into consideration, while trying to keep the modifications to the standard protocol to a minimum. This approach adds fault tolerant capabilities to a normal Boundary Clock as well as the ability to detect attacks on the time source (GPS receiver) directly connected to this Boundary Clock.

The solution managed to satisfy the time synchronization requirements and as an added benefit it also mitigates all types of tested attacks on the primary network time source (GPS receivers). In the case in which an attack is detected, another boundary clock starts acting as the new time source for the current one while the synchronization to the current GPS receiver time source is disabled.

The implementation of the solution was made on open-source code, namely the ptpd2 project, which is a software for version 2 of the PTP standard. Various Linux PCs were used to test and evaluate the solution. In terms of network conditions we tried to emulate a smart grid the best we could, however there are areas where our testbed differs from a real world test case scenario. Some aspects of the testbed network are better than in a real world smart grid (e.g., the geographical distance between nodes) while others are worse (e.g., number of nodes producing traffic and amount of general network traffic).

The results show that this solution is successful at mitigating all types of attacks as

long as our requirements are met.

In the case in which the Grandmaster Clock crashes, a recovery time of 6 seconds was observed. After this, a new time source started being used. The second test case was an attack on the time information provided by the Grandmaster Clock. Our solution managed to detect and recover from this attack in ≈ 2 seconds. We managed to observe an increase in the clock offset after synchronizing to the new time source. This is representative of a real world scenario. Lastly, we verified that delaying network packets coming from the Grandmaster Clock is also detected. Each delayed packet is detected individually and a threshold is put on the maximum number of packets that can be delayed before the Grandmaster Clock is considered failed. The recovery time in this case is equal to the aforementioned threshold. In our case, this parameter was set to 3, translating into a recovery time of 3 seconds.

Further research can be performed to see how the protocol behaves in a real-world smart grid environment, as well as the impact it has on specific parts of the network while under attack.

Bibliography

[1] United State Government Accountability Office, "Electricity grid modernization", May 2008. `http://www.gao.gov/new.items/d11117.pdf`.

[2] A. Meliopoulos, G. Cokkinides, F. Galvan, and B. Fardanesh, "GPS-Synchronized Data Acquisition: Technology Assessment and Research Issues," in *Proceedings of the 39th Annual Hawaii International Conference on System Sciences*, vol. 10, pp. 244c–244c, January 2006.

[3] A. Reis, A. Barros, K. Lenzi, L. Meloni, and S. Barbin, "Introduction to the Software-defined Radio Approach," *IEEE Latin America Transactions*, vol. 10, pp. 1156–1161, January 2012.

[4] C. Fernández–Prades, J. Arribas, P. Closas, C. Avilés, and L. Esteve, "GNSS-SDR: An open source tool for researchers and developers," in *Proceedings of the International Technical Meeting of the Satellite Division of The Institute of Navigation Conference*, September 2011.

[5] J. Aweya and N. Al Sindi, "Role of Time Synchronization in Power System Automation and Smart Grids," in *Proceedings of the IEEE International Conference on Industrial Technology*, pp. 1392–1397, February 2013.

[6] D. L. Mills, "Network Time Protocol Version (NTP)", September 1985.

[7] "IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control System," *IEEE Std 1588-2002*, 2002.

[8] D. L. Mills, "Network Time Protocol Version 4: Protocol and Algorithm Specifications", June 2010.

[9] "IEEE Standard for Synchrophasor Measurements for Power Systems," *IEEE Std. C37.118.2011*, December 2011.

[10] "Precision Time Protocol Daemon - ptpd2", 2011. http://ptpd2.sourceforge.net.

[11] "Code division multiple access". https://en.wikipedia.org/wiki/Code_division_multiple_access.

[12] "Global Positioning System standard positioning service performance standard 4th Edition", September 2008.

[13] "Interface Specification IS-GPS-200", September 2013.

[14] H. Hu and N. Wei, "A study of GPS jamming and anti-jamming," in *Proceedings of the 2nd International Conference on Power Electronics and Intelligent Transportation System*, vol. 1, pp. 388–391, December 2009.

[15] D. Borio, C. O'Driscoll, and J. Fortuny, "Jammer impact on Galileo and GPS receivers," in *Proceedings of the International Conference on Localization and GNSS*, pp. 1–6, June 2013.

[16] The Economist, "GPS jamming, Out of sight", July 2013. http://www.economist.com/news/international/21582288-satellite-positioning-data-are-vitalbut-signal-surprisingly-easy-disrupt-out.

[17] J. Larcom and H. Liu, "Modeling and characterization of GPS spoofing," in *Proceedings of the IEEE International Conference on Technologies for Homeland Security*, pp. 729–734, November 2013.

[18] T. Humphreys, B. Ledvina, M. Psiaki, B. O'Hanlon, Kintner, and M. Paul Jr., "Assessing the Spoofing Threat: Development of a Portable GPS Civilian Spoofer," in *Proceedings of the 21st International Technical Meeting of the Satellite Division of The Institute of Navigation*, pp. 2314–2325, September 2008.

[19] Y. Fan, Z. Zhang, M. Trinkle, A. Dimitrovski, J. Song, and H. Li, "A Cross-Layer Defense Mechanism Against GPS Spoofing Attacks on PMUs in Smart Grids," *IEEE Transactions on Smart Grid*, vol. PP, no. 99, pp. 1–1, 2014.

[20] Z. Zhang, M. Trinkle, L. Qian, and H. Li, "Quickest detection of GPS spoofing attack," in *Proceedings of the Military Communications Conference*, pp. 1–6, October 2012.

[21] T. Nighswander, B. Ledvina, J. Diamond, R. Brumley, and D. Brumley, "GPS Software Attacks," in *Proceedings of the ACM Conference on Computer and Communications Security*, pp. 450–461, 2012.

[22] "IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems," *IEEE Std 1588-2008 (Revision of IEEE Std 1588-2002)*, pp. c1–269, July 2008.

[23] Calix Inc., "Asymmetric Networks – Timing Solutions," 2010. `http://www.chronos.co.uk/files/pdfs/itsf/2010/Day3/02-Asymmetric_Network_Timing_Solutions.pdf`.

[24] J. Jasperneite, K. Shehab, and K. Weber, "Enhancements to the time synchronization standard IEEE 1588 for a system of cascaded bridges," in *Proceedings of the IEEE International Workshop on Factory Communication Systems*, pp. 239–244, September 2004.

[25] T. Mizrahi, "Time synchronization security using IPsec and MACsec," in *Proceedings of the International IEEE Symposium on Precision Clock Synchronization for Measurement Control and Communication*, pp. 38–43, September 2011.

[26] P. Estrela and L. Bonebakker, "Challenges deploying PTPv2 in a global financial company," in *Proceedings of the International IEEE Symposium on Precision Clock Synchronization for Measurement Control and Communication*, pp. 1–6, September 2012.

[27] L. Lamport, R. Shostak, and M. Pease, "The Byzantine Generals Problem," *in Proceedings of the ACM Transactions Programming Languages and Systems*, vol. 4, no. 3, pp. 382–401, 1982.

[28] X. Cheng and L. Zhang, "A research of inter-process communication based on shared memory and address-mapping," in *Proceedings of the International Conference on Computer Science and Network Technology*, vol. 1, pp. 111–114, December 2011.

[29] K. Correll, N. Barendt, and M. Branicky, "Design Considerations for Software Only Implementations of the IEEE 1588 Precision Time Protocol," 2005.

[30] "TC Linux User's Manual", December 2001. `http://lartc.org/manpages/tc.txt`.

[31] J. Tournier and O. Goerlitz, "Strategies to secure the IEEE 1588 protocol in digital substation automation," in *Proceedings of the 4th International Conference on Critical Infrastructures*, pp. 1–8, March 2009.

[32] R. Onica, N. Neves, and A. Casimiro, "Fault-Tolerant Precision Time Protocol for Smart Grids," in *Proceedings of the 7th National INForum Informatics Symposium*, July 2015.