Capítulo 2

Confiabilidade e Segurança nos Sistemas Distribuídos Físico-Digitais

Raimundo J. de A. Macêdo[1], Alirio S. de Sá[1], Allan E. S. Freitas[2], Paulo E. Veríssimo[3], Sérgio Gorender[1], Margarete O. S. de Sá[1]

[1]Universidade Federal da Bahia, Brasil
[2]Instituto Federal de Educação, Ciência e Tecnologia da Bahia, Brasil
[3]King Abdullah University of Science and Technology, Reino da Arábia Saudita

Abstract

Distributed cyber-physical systems, also referred to as physical-digital systems, are characterized by geographically dispersed physical components interconnected through communication networks, thereby creating a decentralized and collaborative infrastructure. These systems encompass a wide range of devices such as temperature sensors, industrial robots, biosensors, drones, cameras, and other devices that interact with the physical environment. They find application across various industries including manufacturing, transportation, healthcare, environmental control, entertainment, commerce, and security. However, the physical-digital relationship in these scenarios introduces significant challenges such as interoperability, reliability, cybersecurity, and scalability. To address these challenges, the utilization of autonomic management in these systems proves beneficial. Autonomic management enables the systems to self-monitor their states and make decisions based on the collected data. This text aims to present a theoretical-practical approach for constructing autonomous or autonomic distributed systems that can fulfill the performance, reliability, and security requirements of distributed physical-digital systems. Understanding the increasing complexity of these systems and effectively managing them is crucial to ensuring their efficiency and effectiveness.

Resumo

Sistemas ciberfísicos distribuídos, também conhecidos como sistemas físico-digitais, são caracterizados por componentes físicos geograficamente dispersos e interconectados por meio de redes de comunicação, criando assim uma infraestrutura descentralizada e colaborativa. Esses sistemas abrangem uma ampla variedade de dispositivos, como sensores de

temperatura, robôs industriais, biossensores, drones, câmeras e outros dispositivos que interagem com o ambiente físico. Eles encontram aplicação em diversas indústrias, incluindo manufatura, transporte, saúde, controle ambiental, entretenimento, comércio e segurança. No entanto, a relação físico-digital nesses cenários apresenta desafios significativos, como interoperabilidade, confiabilidade, cibersegurança e escalabilidade. Para enfrentar esses desafios, a utilização do gerenciamento autonômico nesses sistemas se mostra benéfica. O gerenciamento autonômico permite que os sistemas monitorem seus próprios estados e tomem decisões com base nos dados coletados. Este texto tem como objetivo apresentar uma abordagem teórico-prática para a construção de sistemas distribuídos autônomos ou autonômicos capazes de atender aos requisitos de desempenho, confiabilidade e segurança dos sistemas físico-digitais distribuídos. Compreender a crescente complexidade desses sistemas e gerenciá-los de forma eficaz é crucial para garantir sua eficiência e eficácia.

2.1. Contexto e Motivações

2.1.1. Introdução

Sistemas físico-digitais distribuídos, também conhecidos como sistemas ciberfísicos distribuídos (do inglês, *cyber-physical systems*) são aqueles em que os componentes físicos estão geograficamente distribuídos e conectados por meio de redes de comunicação, formando uma infraestrutura descentralizada e colaborativa. Tais sistemas abrangem uma ampla variedade de dispositivos, como sensores de temperatura, robôs industriais, biossensores, drones, câmeras e outros dispositivos de interação com o ambiente físico. Essas características estão presentes em ambientes como os da indústria 4.0, da indústria 5.0, de realidade virtual e aumentada, de metaverso, entre outros, com aplicação em diversos setores estratégicos, incluindo manufatura, transporte, saúde, controle ambiental, entretenimento, comércio e segurança. Utilizaremos os termos "físico-digital" e "ciberfísico" de forma intercambiável no decorrer do texto.

Além dos requisitos dos sistemas distribuídos convencionais, como interoperabilidade, confiabilidade, segurança cibernética e escalabilidade, a relação físico-digital apresenta desafios adicionais devido à complexa interação entre o sistema computacional e os processos físicos subjacentes. Dessa forma, os sistemas físico-digitais envolvem o projeto seguro e a operação temporalmente previsível de sistemas de computação que interagem com o ambiente [Kopetz 1997; Macêdo et al. 2004], com sofisticação cada vez maior, incluindo: infraestruturas de controle de computador; monitoramento e operações de saúde; veículos autônomos em terra, ar e espaço; sistemas robóticos; monitoramento domiciliar e ambiental; etc. O projeto desses sistemas exige os fundamentos teóricos combinados de distribuição, tempo real e computação embarcada, lidando, nas mais complexas dessas infraestruturas, com um conflito frequente entre complexidade, imprevisibilidade e a necessidade de operação. Além disso, a capacidade de tornar essas propriedades sustentáveis ao longo do tempo, sob cenários de ameaças persistentes e/ou em evolução, é um grande desafio. Nesses sistemas, a operação em tempo real é fundamental, não há como desligar ou fechar, para obstar um ataque de segurança. Essa lacuna é agravada pela convergência dos Sistemas físico-digitais com o complexo Internet-Nuvem-Web.

As consequências de falhas operacionais ou ataques à segurança podem ser graves em alguns cenários, em particular porque os sistemas físico-digitais muitas vezes operam em proximidade com humanos, podendo assim causar danos físicos, incluindo risco de vida. Nessas condições, definitivamente são necessários sistemas resilientes e dinamicamente

adaptáveis a novas condições operacionais. Por exemplo, um sistema computacional encarregado de controlar uma rede ferroviária deve obedecer a restrições temporais precisas. Além disso, em tais cenários, há não apenas a interação com elementos físicos, mas também a integração de processos computacionais externos, abrangendo diversos domínios, desde a Internet das Coisas até a Computação em Nuvem e a participação humana.

Para lidar com tais desafios, paradigmas de proteção baseados em segurança clássica ou confiabilidade clássica, embora necessários, são insuficientes se trabalharem isoladamente e de forma rígida. Muitas falhas induzidas por usuários de forma maliciosa ou acidental levam à compreensão da necessidade de tratar simultaneamente falhas acidentais, erros de projeto e condições operacionais inesperadas, aumentando a robustez e resiliência desses sistemas. Nesse contexto, busca-se alcançar a resiliência dos sistemas computacionais, proporcionando uma forma autonômica de gerenciá-los de maneira resistente a ameaças diversas, sejam acidentais ou maliciosas. Nessa perspectiva, os sistemas devem possuir defesas incorporadas desde os princípios iniciais, que sejam capazes de lidar com diferentes tipos de ameaças e possam ser configurados de forma incremental. Além disso, esses sistemas devem responder automaticamente a ameaças, tolerar falhas e intrusões, adaptar-se à gravidade das situações e fornecer operação autônoma e sustentável ao longo de sua vida, com capacidade de recuperação e autocorreção.

Para lidar com esses desafios, em tempos e na qualidade desejada, uma abordagem frequentemente utilizada é a gestão autônoma ou autonômica, na qual os sistemas são configurados para monitorar seus próprios estados e tomar decisões com base em dados coletados, visando atender aos requisitos funcionais e de qualidade de serviço dinamicamente estabelecidos pelo ambiente operacional ou dos utilizadores de tais sistemas [Macêdo 2008b; Kumar et al. 2017]. Ademais, dada a grande complexidade de sistemas físico-digitais em larga escala, além da desejada autonomia, os problemas de confiabilidade e segurança são tratados em uma perspectiva unificadora e integrada [Veríssimo et al. 2009].

Para melhor compreensão da literatura sobre confiabilidade e segurança nos sistemas distribuídos físico-digitais, apresentaremos, resumidamente, um panorama da literatura clássica sobre esses conceitos, o que ajudará no percurso das demais seções.

2.1.2. Modelos de Sistemas Distribuídos

O campo dos sistemas distribuídos iniciou-se na década de 1970 com o surgimento das primeiras redes. A ideia fundamental era a de que computadores distintos poderiam colaborar e coordenar-se para atingir objetivos específicos, compartilhando recursos entre processos distribuídos. A pesquisa nesse campo pode ser dividida em dois subcampos principais: modelos de computabilidade e algoritmos básicos, por um lado, e infraestrutura e ferramentas de software, por outro. Os modelos de computabilidade tratam da resolução de problemas fundamentais e recorrentes de computação distribuída (por exemplo, consenso, exclusão mútua, etc.) em modelos específicos. A infraestrutura e as ferramentas de software abordam os problemas de como desenvolver e implantar sistemas a partir de processos distribuídos, o que envolve padrões de comunicação, como chamadas de procedimento remoto (RPC), disponibilização de serviços, protocolos padrão para comunicação e descoberta de objetos/serviços, segurança, e assim por diante.

Um sistema distribuído (SD) é composto por um conjunto Π de processos que se comunicam e sincronizam entre si, trocando mensagens por meio de canais de comunicação. Essa definição é geralmente ampliada para incluir especificações sobre o comportamento dos

processos e canais, abordando garantias temporais das operações e padrões de falha. A pesquisa no subcampo de computabilidade e algoritmos básicos concentra-se, em geral, na análise da computabilidade e complexidade dos problemas fundamentais e recorrentes encontrados em sistemas distribuídos, abrangendo uma variedade de modelos de SD. Por exemplo, podemos ter um modelo de sistema distribuído onde cada par de processos está conectado por um canal bidirecional assíncrono e confiável. Isso significa que esses canais não criam, modificam ou perdem mensagens, e as mensagens podem levar um tempo arbitrário e ilimitado para serem entregues. Esse modelo é frequentemente chamado de tempo-livre ou assíncrono, e foi comprovado que não existe uma solução determinística para o problema de consenso nesse modelo [Fisher et al. 1985]. O problema de consenso, que se refere a acordos entre processos distribuídos sobre determinados estados, é recorrente e serve de bloco de construção para outros problemas [Hurfin et al. 1999]. Por outro lado, em sistemas síncronos, onde os atrasos de transmissão de mensagens e execução de processos são limitados, vários problemas relacionados à computação distribuída, como consenso e *membership*, foram resolvidos [Lamport et al. 1982; Cristian 1991]. A má notícia é que as suposições do sistema síncrono são válidas apenas em cenários restritos. Por exemplo, essas suposições não se aplicam à Internet.

Em muitos casos, no entanto, a escolha entre um modelo síncrono ou um modelo assíncrono pode não ser a melhor opção. Sistemas reais frequentemente apresentam comportamentos síncronos em partes específicas ou em certas condições de operação. Isso tem levado ao desenvolvimento de modelos intermediários, ou híbridos, que se situam entre os modelos síncronos e assíncronos, nos quais foi provado que muitos problemas clássicos de sistemas distribuídos, como consenso, têm solução [Dwork et al. 1988; Chandra e Toueg 1996; Veríssimo e Casimiro 2002; Gorender e Macêdo 2002; Gorender, Macêdo e Raynal 2005; Macêdo, Gorender e Cunha 2005; Veríssimo 2006; Macêdo e Gorender 2008; Macêdo e Gorender 2012].

Alguns desses modelos híbridos assumem comportamentos alternados ao longo do tempo, como o modelo de sincronia temporal proposto por [Cristian e Fetzer 1999] e o modelo assíncrono com detectores de defeitos desenvolvido por [Chandra e Toueg 1996]. Outros modelos intermediários permitem composições heterogêneas, como o modelo TCB [Veríssimo e Casimiro 2002] e o modelo Síncrono Particionado [Macêdo e Gorender 2008; Macêdo e Gorender 2009; Macêdo e Gorender 2012]. Esses modelos híbridos heterogêneos, que combinam aspectos síncronos e assíncronos, têm um maior potencial de se adequarem aos sistemas físico-digitais, pois interações físicas requerem respostas em tempos pré-estabelecidos, enquanto as interações em redes de larga escala são geralmente assíncronas.

2.1.3. Confiabilidade e conceitos relacionados

Confiabilidade (*reliability*) e disponibilidade (*availability*) são propriedades complementares que qualificam a confiança que pode ser depositada no funcionamento correto de um sistema, também conhecida como *dependabilidade* (*dependability*) do sistema [Jalot 1994; Kopetz 1997; Avizienis et al. 2004]. A confiabilidade está relacionada à capacidade do sistema de se manter operacional, fornecendo continuamente um serviço correto, enquanto a disponibilidade refere-se à capacidade do sistema de estar disponível, alternando entre os estados operacional e não operacional. Para lidar com a ocorrência de falhas, se mantendo operacional, o sistema precisa dispor de mecanismos de tolerância a falhas. Como não é

possível controlar todos os fatores que levam a falhas, a confiabilidade e a disponibilidade são melhor expressas em termos probabilísticos [Cristian 1991; Veríssimo e Rodrigues 2000].

O mau funcionamento do sistema distribuído físico-digital pode implicar em prejuízos financeiros, ambientais, ou até mesmo perda de vidas humanas. Por melhores que sejam as técnicas usadas na construção destes sistemas, diversos fatores podem levar os mesmos a apresentarem defeitos: desgastes físicos de componentes, falhas humanas, fenômenos e/ou desastres naturais etc. Portanto, mecanismos adequados de tolerância a falhas devem ser empregados em tais sistemas. Os termos *falha*, *erro*, *defeito*, adotados neste texto, seguem o padrão de terminologia amplamente adotado pela comunidade científica internacional da área de tolerância a falhas [Avizienis et al. 2004]; sendo que *fault*, *error* e *failure* são traduzidos, respectivamente, para falha, erro e defeito.

A semântica de falhas [Cristian 1991] está relacionada ao comportamento de componentes ou subsistemas que apresentam falhas. Esse comportamento é influenciado pelo ambiente computacional e de comunicação subjacente. Em particular, as garantias temporais de operação desempenham um papel crucial. Como apresentado anteriormente, em modelos síncronos, há garantias temporais nas operações internas aos processos e na comunicação entre processos. Por outro lado, em modelos assíncronos, não há garantias temporais. A escolha da técnica de tolerância a falhas depende da semântica de falhas e é implementada por meio de redundância de componentes, estruturas ou procedimentos.

Conceitualmente, um defeito denota um desvio entre o comportamento de um serviço e o que foi especificado para tal serviço [Avizienis et al. 2004]. Tal desvio é originado de um estado errôneo (i.e., erro) ao qual é levado o sistema na presença de uma falha. Um erro pode permanecer latente até que algum evento no sistema promova a sua ativação. Como a falha é um evento indesejado que pode inserir um erro no sistema, pode-se afirmar que a falha é a causa indireta e primária do defeito – i.e., o evento de falha causa o estado de erro que, quando ativado, leva a um serviço defeituoso. Ou seja, um erro pode permanecer latente até que algum evento no sistema o ative.

Os conceitos de defeito e falha são relativizados pela organização hierárquica dos sistemas. De modo geral, um sistema é composto por outros sistemas (ditos subsistemas ou componentes). Da mesma forma, estes subsistemas são estruturados a partir de outros componentes e assim por diante, até se alcançar os subsistemas mais elementares, para os quais esta subdivisão não é evidente ou não tem um sentido próprio. O desvio no serviço provido por um componente ou subsistema (i.e., defeito) representa uma falha para o sistema ao qual tal componente está inserido, conforme ilustrado na Figura 2.1. Se a respectiva falha não for devidamente tratada, o erro gerado no estado do sistema pode eventualmente fazer com que o respectivo serviço esteja fora de sua especificação (defeito).

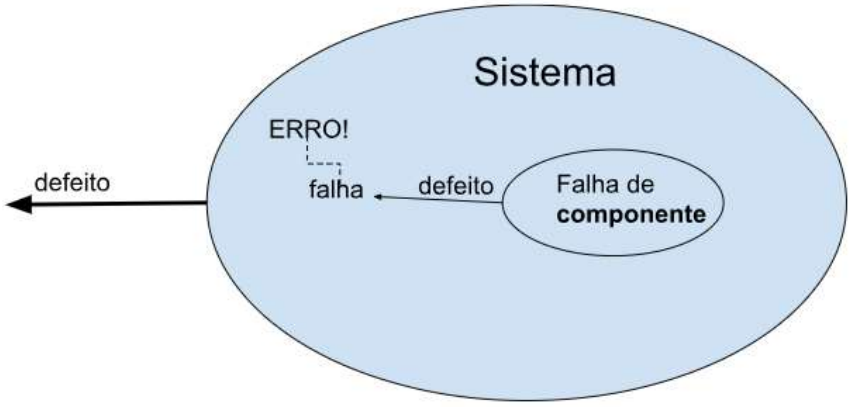

Figura 2.1: Hierarquia falha-erro-defeito

2.1.3.1. Classificação das falhas

A tipificação de falhas é feita considerando como as falhas afetam o comportamento dos componentes do sistema. Existem várias formas de classificar as falhas [Avizienis et al. 2004; Jalote 1994]. Na literatura técnica relacionada, são comumente utilizadas as seguintes classes de falhas:

- **Falha silenciosa** – ocorre quando o componente para de funcionar, deixando de responder a qualquer requisição de serviço. Esse é o tipo de falha mais comum utilizada em projetos de sistemas distribuídos. Nos sistemas distribuídos caracterizados como síncronos, essas falhas podem ser detectadas e recebem o nome de falhas do tipo *fail-stop*; nos modelos assíncronos, essas falhas são apenas estimadas e são denominadas falhas do tipo *crash*.
- **Falha por omissão** – ocorre quando o componente omite (ou deixa de produzir) o resultado esperado, conforme sua especificação. Observe que a falha por parada é um tipo particular de falha por omissão, na qual o componente omite permanentemente os resultados esperados.
- **Falha temporal ou de desempenho** – ocorre quando o componente responde fora do prazo especificado. Observe que a falha por omissão pode ser caracterizada como uma falha temporal, na qual o resultado esperado nunca será produzido dentro do prazo, caso este exista. Falhas temporais fazem somente sentido em sistemas distribuídos síncronos, onde os tempos de execução de processos e transmissão de mensagens são conhecidos.
- **Falha de valor** – ocorre quando o componente produz uma saída incorreta para um dado valor de entrada.
- **Falha arbitrária ou bizantina** – ocorre quando um componente pode manifestar qualquer tipo de comportamento na presença de falha, podendo parar de funcionar, omitir resultados, produzir resultados fora do prazo ou ainda produzir valores incorretos. A Figura 2.2 ilustra a hierarquia de classes apresentadas acima.

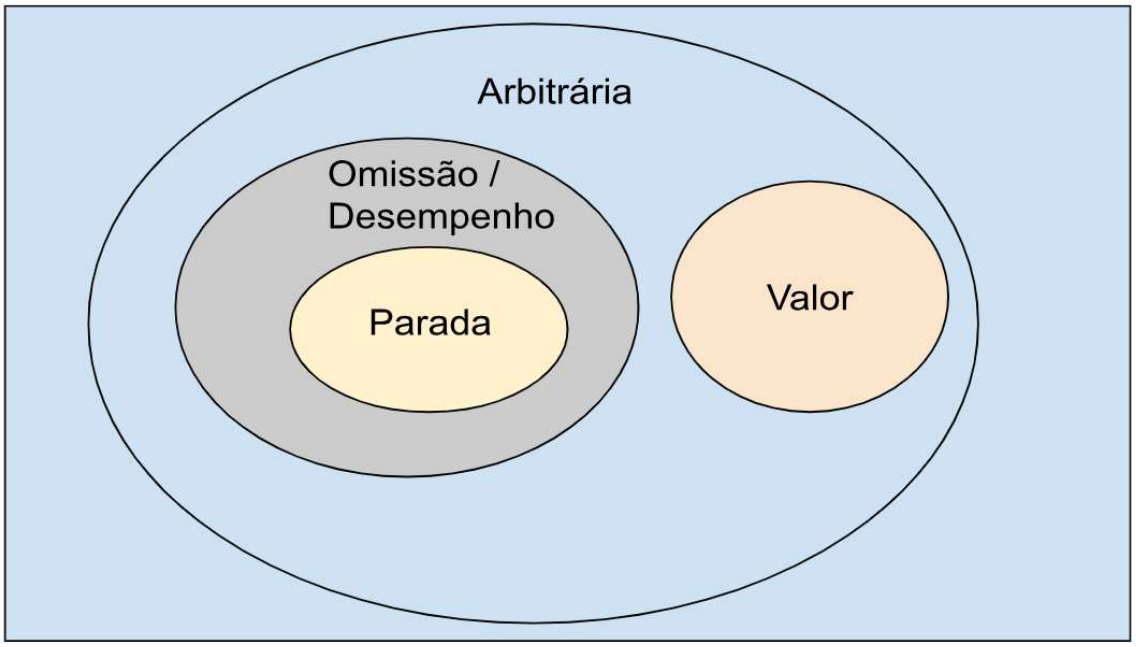

Figura 2.2. Classes de falha

2.1.4. Segurança

A segurança em sistemas distribuídos físicos-digitais é de extrema importância para garantir a integridade, confidencialidade e disponibilidade da infraestrutura crítica e dos dados sensíveis. Esses sistemas combinam elementos físicos com recursos computacionais, criando um ambiente complexo de rede onde o impacto de violações de segurança pode ter consequências graves.

A natureza desses sistemas introduz desafios de segurança únicos. Os ataques podem visar tanto os componentes cibernéticos quanto físicos, visando interromper as operações, comprometer a integridade dos dados ou obter acesso não autorizado. Portanto, uma abordagem abrangente de segurança é necessária para lidar com essas preocupações.

Um aspecto fundamental para garantir a segurança em sistemas distribuídos físico-digitais é a implementação de mecanismos robustos de autenticação e controle de acesso. Autenticação forte de usuário, protocolos de comunicação seguros e permissões de acesso detalhadas são essenciais para evitar que indivíduos não autorizados assumam o controle de componentes críticos ou acessem informações sensíveis.

A integridade e a confidencialidade dos dados são fundamentais para proteger o sistema contra modificações não autorizadas ou violações de dados. Criptografia, armazenamento seguro e protocolos de comunicação seguros desempenham um papel vital na proteção de dados sensíveis contra interceptação, adulteração ou divulgação não autorizada.

Outra consideração crítica é a resiliência do sistema contra vários tipos de ataques. Sistemas de detecção de intrusões, algoritmos de detecção de anomalias e mecanismos de monitoramento de segurança ajudam a identificar e mitigar ameaças potenciais em tempo real. Além disso, a implementação de estratégias de redundância, tolerância a falhas e recuperação de desastres pode garantir a disponibilidade do sistema e uma resposta rápida a incidentes de segurança.

A realização de avaliações de segurança regulares, testes de vulnerabilidade e atualizações é crucial para manter a segurança dos sistemas distribuídos físicos-digitais. À medida que surgem novas vulnerabilidades e técnicas de ataque evoluem, é essencial estar atualizado com as melhores práticas de segurança e aplicar as devidas atualizações para mitigar riscos potenciais.

Em suma, a segurança em sistemas distribuídos físico-digitais requer uma abordagem holística e proativa. Ao integrar medidas de segurança robustas, monitoramento contínuo e avaliações regulares, as organizações podem melhorar a resiliência e a confiabilidade desses sistemas críticos, protegendo-os contra ameaças potenciais e garantindo seu funcionamento confiável no mundo interconectado de hoje.

2.2. Gestão autonômica de sistemas distribuídos

2.2.1. Sistemas autogerenciávies

Nesta seção, descreveremos as motivações que levaram pesquisadores de computação a proporem estruturas autogerenciáveis, ou autonômicas, para lidar com a complexidade crescente dos sistemas computacionais distribuídos, sobretudo quando se consideram sistemas de sistemas, computação em nuvem, Internet das Coisas, sistemas físicos-digitais e, mais recentemente, sistemas da Indústria 4.0 e da Indústria 5.0. O termo autonômico será precisamente definido e contextualizado para os sistemas distribuídos; todavia, para melhor entendimento da literatura sobre o tema, apresentaremos resumidamente para o leitor os esforços e terminologias utilizadas em diversas áreas da computação onde a propriedade de autonomia é tratada de forma semelhante, ainda que com enfoques variados e/ou em camadas distintas dos sistemas computacionais: engenharia de software, sistemas de tempo real, sistemas robóticos, gerência de redes, entre outras áreas.

A Iniciativa de Computação Autônoma da IBM (ACI) visa o desenvolvimento de sistemas computacionais capazes de autogerenciamento, tomando decisões autonomamente a partir de políticas de alto nível, caracterizados por propriedades de auto-*, como autoconfiguração, autorregeneração, autootimização, autoproteção entre outras [Horn 2001]. Desde o lançamento da iniciativa em 2001, grandes esforços de pesquisa têm se concentrado em como projetar tais sistemas e em mecanismos para as propriedades de auto-*. É senso comum na literatura que, para serem autônomos, os sistemas devem ser compostos por uma hierarquia de subsistemas autônomos, desde o nível mais alto (nível do usuário) até os componentes básicos mais baixos.

A ideia de autogerenciamento ou autonomia é muito antiga. De fato, até mesmo o próprio conceito de execução automática de programas de computador em arquiteturas Von Neumann traz a ideia de execução autônoma no nível da máquina. Além disso, o conceito de autonomia tem sido explorado em muitos campos, como inteligência artificial, robótica, engenharia de software, gerenciamento de redes, automação e sistemas de controle, computação biológica, confiabilidade, etc. No entanto, as principais contribuições da ACI parecem ser a perspectiva de avaliar o quão autônomo (ou autônoma) um sistema pode ser - por meio da proposição de níveis de maturidade autônoma - e também de disseminar a necessidade urgente de sistemas mais autônomos para lidar com a complexidade cada vez maior de gerenciamento dos sistemas e aplicativos de computação modernos. Paradigmas como a computação em nuvem, que permitem maior flexibilidade para os usuários finais em termos de alocação de recursos e composições de serviços, têm reforçado a necessidade de

sistemas autônomos. O fato de os usuários não precisarem planejar antecipadamente o provisionamento de recursos e os recursos computacionais (como largura de banda de rede, CPU, etc.) serem alocados sob demanda para atender aos acordos de nível de serviço definidos pelo usuário torna esses novos paradigmas muito atrativos. No entanto, se adaptar à variabilidade de recursos e às mudanças nos requisitos do usuário em tempo de execução é um grande desafio. Como consequência, esses sistemas precisam ter a capacidade de serem autogerenciáveis, se reconfigurando e se ajustando continuamente para atingir certos objetivos.

A seguir apresentaremos uma visão dos sistemas distribuídos, a partir da literatura clássica em uma perspectiva autonômica, como apresentado em [Macêdo 2013].

2.2.2. Sistemas Distribuídos Autonômicos

As suposições clássicas em modelos de sistemas distribuídos não levam em consideração a possibilidade de mudanças ao longo do tempo, o que representa uma limitação para sistemas autonômicos, nos quais as características do sistema de computação subjacente podem se alterar dinamicamente. Portanto, argumentamos que um sistema distribuído autonômico deve estar ciente da validade de suas próprias suposições, seguindo as garantias verificadas de qualidade de serviço e permitindo que os algoritmos se adaptem de acordo. Por exemplo, aproveitando comportamentos síncronos percebidos, quando possível, ou alternando para soluções síncronas quando as suposições relacionadas não forem mais válidas.

A evolução de um sistema distribuído frequentemente envolve se adaptar a novos requisitos de aplicação ou mudanças no ambiente de computação, como falhas ou adição/remoção de recursos computacionais. Portanto, ser adaptativo é um requisito fundamental para lidar com a natureza dinâmica desses sistemas. Consequentemente, sistemas adaptativos têm recebido grande atenção em áreas como confiabilidade, sistemas de tempo real e engenharia de software. Por exemplo, um sistema tolerante a falhas é inerentemente adaptativo, pois pode continuar funcionando corretamente mesmo diante de falhas de componentes, demonstrando adaptabilidade em relação a falhas de tais componentes.

É possível projetar sistemas que podem modificar seus mecanismos de tolerância a falhas durante a execução para se adequar a novas condições operacionais ou otimizar aspectos de desempenho, como sobrecarga de mensagens. Para compreender a natureza autonômica de um sistema distribuído em termos de sua adaptabilidade, classificamos os sistemas distribuídos autonômicos com base no grau de controle que podem exercer sobre os mecanismos adaptativos, se aplicável, da seguinte forma, conforme proposto por [Macêdo 2012]:

1. **Não adaptativo**: não são fornecidas capacidades de adaptação.
2. **Adaptativo offline**: a adaptação é possível, mas apenas antes da execução, sem adaptação em tempo de execução.
3. **Adaptativo online**: a adaptação ocorre em tempo de execução com base em objetivos predefinidos, mas as políticas não podem ser controladas durante a execução.
4. **Adaptativo autonômico**: as políticas ou objetivos de adaptação podem ser modificados em tempo de execução.

Para permitir a implantação de sistemas distribuídos autonômicos, os blocos fundamentais de construção dos sistemas distribuídos devem fornecer a capacidade para que

as aplicações ou serviços de camada superior possam controlar seu comportamento com base em políticas, ou objetivos específicos. Essas políticas ou objetivos podem definir requisitos de qualidade de serviço (QoS), restrições do ambiente computacional ou uma combinação de ambos. Um aspecto crucial do projeto é estabelecer objetivos ou derivá-los de acordos de nível de serviço (SLAs) de níveis superiores. Nós aplicamos essa abordagem geral para implementar detectores de defeitos [de Sá e Macêdo 2010], comunicação em grupo [Macêdo et al. 2013] e replicação bizantina [de Sá et al. 2013], que são blocos de construção essenciais em sistemas distribuídos tolerantes a falhas.

Por exemplo, no caso dos detectores de defeitos, o sistema de camada superior, seja um usuário humano ou outro sistema, pode controlar o QoS desejado para a detecção de defeitos dentro dos recursos disponíveis, como tempo de detecção e precisão. Nesse caso, temos um comportamento autonômico adaptativo, pois o objetivo autonômico pode ser alterado dinamicamente. A adaptação online representa uma forma mais fraca de sistema distribuído autonômico. É importante observar que a definição de tais objetivos autonômicos ou políticas de adaptabilidade, depende do entendimento da dinâmica dos blocos de construção de sistemas distribuídos relacionados e respectivos compromissos de desempenho.

No desenvolvimento de sistemas distribuídos físico-digitais, embora os mecanismos adaptativos sejam eficazes para garantir confiabilidade e segurança, esses mecanismos também podem precisar ser modificados para lidar com mudanças imprevistas no ambiente computacional e/ou requisitos de clientes/sistemas. Essas mudanças podem incluir novas configurações do sistema, atualizações nos componentes do sistema ou até mesmo novos acordos de nível de serviço (SLAs). Portanto, muitas vezes, a adaptação deve ocorrer em tempo real, sem conhecimento prévio completo ou antecipação de eventos potenciais.

Para lidar com esses desafios, o sistema precisa possuir comportamento autogerenciável ou autonômico em sua capacidade máxima (i.e., adaptativo autonômico). Isso significa que o sistema deve ser capaz de alterar dinamicamente suas políticas de adaptação em tempo de execução, conforme necessário. Portanto, os laços de retroalimentação são essenciais para detectar tanto o ambiente quanto os requisitos do sistema. Os sistemas devem se adaptar dinamicamente com base nessas condições percebidas e em políticas de nível superior.

Para projetar um protocolo autonômico desse tipo, é necessário abordar cuidadosamente várias questões. Entre elas, destacam-se: quais devem ser os objetivos de desempenho do protocolo e como eles podem ser expressos de maneira adequada? Como as dinâmicas do sistema e do protocolo podem ser modeladas de forma eficaz dentro dos laços de retroalimentação? Em quais momentos e de que forma o sistema deve se adaptar a diferentes modos de operação? E, por fim, com qual frequência os componentes do sistema e as variáveis do protocolo devem ser monitorados e reconfigurados?

Em geral, cada protocolo autonômico é modelado como um laço de controle realimentado, como ilustrado na Figura 2.3.

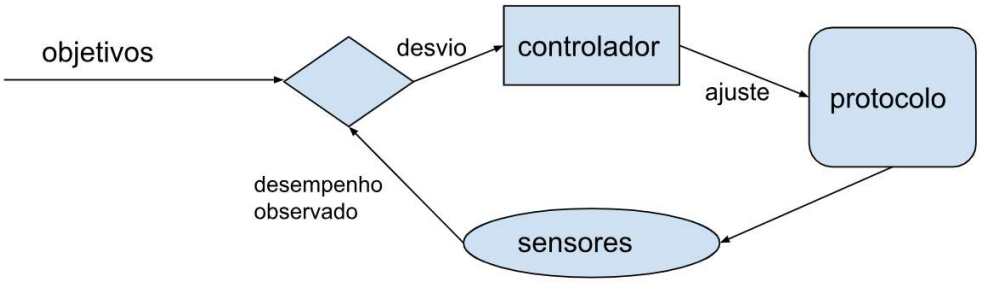

Figura 2.3. Protocolo autonômico.

O desempenho do protocolo, observado por meio de variáveis operacionais, denominadas na Figura 2.3 de sensores, é comparado com o desempenho declarado nos objetivos. Havendo desvios entre o desejado e observado, o controlador promove ajustes sobre mecanismos ou variáveis do protocolo que está sendo controlado. Quando os objetivos não mudam durante a execução, temos um protocolo **adaptativo online**; se os objetivos podem ser modificados em tempo de execução, temos um protocolo **adaptativo autonômico.**

2.3. Implementação de Sistemas distribuídos físico-digitais confiáveis e seguros

Esta seção apresenta aspectos básicos relacionados à implementação de sistemas distribuídos autonômicos, considerando a incorporação de habilidades de gestão autonômica em mecanismos e protocolos básicos usados na construção dos sistemas distribuídos. Como mencionado anteriormente, o projeto desses sistemas precisa conciliar requisitos da previsibilidade temporal, da computação distribuída e, por vezes, do sistema embarcado [Macêdo e Farines 2018].

Primeiramente, é discutida uma visão conceitual que ajuda a entender como mecanismos e protocolos básicos usados em sistemas distribuídos podem ser estendidos para incorporar facilidades de gestão autonômica sem, contudo, violar propriedades básicas de correção – esta visão conceitual, será exemplificada com uma discussão breve, usando protocolos e mecanismos básicos para sistemas distribuídos confiáveis, como protocolos de comunicação em grupo e protocolos de replicação. Em seguida, é discutido como o modelo conceitual é traduzido em desenho arquitetural de software que considera a representação de mecanismos e protocolos básicos em um loop autonômico – este desenho é usado para apresentar como políticas e requisitos definidos pelos usuários/aplicações são usados por tais mecanismos e protocolos autonômicos durante o processo de auto adaptação. O desenho arquitetural também é usado para apresentar de forma didática as atividades básicas do processo de auto adaptação, as quais envolvem o sensoriamento do ambiente distribuído e do comportamento dos próprios mecanismos e protocolos, o planejamento e a realização dos ajustes necessários para o atendimento das políticas e requisitos definidos pelos usuários e aplicações. Por fim, são discutidos de forma mais detalhada os desafios e algumas alternativas de implementação das atividades do laço autonômico envolvendo os protocolos e mecanismos básicos dos sistemas distribuídos.

2.3.1. Aspectos Básicos do Projeto de Sistemas Distribuídos Confiáveis

Para garantir o funcionamento correto e contínuo de sistemas distribuídos, mecanismos de tolerância a falhas são fundamentais [Cristian 1991]. A tolerância a falhas em sistemas distribuídos envolve o projeto de mecanismos e estratégias para garantir que o sistema

continue funcionando corretamente, mesmo na presença de falhas, o que envolve, resumidamente:

1. **Redundância:** uma das principais abordagens para alcançar a tolerância a falhas é por meio da redundância. A redundância envolve a replicação de componentes críticos, dados ou serviços em vários nós do sistema distribuído. Ao ter várias cópias do mesmo componente, se uma cópia falhar ou se tornar indisponível, o sistema pode contar com as cópias redundantes para continuar suas operações. A redundância pode ser implementada em vários níveis, como hardware, software ou dados.
2. **Detecção de defeitos de componentes:** sistemas distribuídos tolerantes a falhas utilizam mecanismos de detecção de defeitos para identificar e localizar falhas em seus componentes. Técnicas como monitoramento ou *checksums* são usadas para detectar anomalias ou desvios de comportamentos esperados. Esses mecanismos monitoram constantemente o sistema e seus componentes para identificar quaisquer falhas ou erros potenciais.
3. **Isolamento de falhas:** uma vez que uma falha ou falha é detectada, os sistemas tolerantes a falhas visam isolar o componente com defeito para evitar a propagação de erros e manter a funcionalidade geral do sistema. Técnicas de isolamento envolvem a duplicação de processos ou recursos, onde componentes redundantes assumem as responsabilidades do componente com defeito. Isolar falhas ajuda a garantir que o restante do sistema possa continuar operando sem ser afetado pelo componente com defeito.
4. **Recuperação de falhas:** após isolar uma falha, os sistemas tolerantes a falhas empregam mecanismos de recuperação para restaurar o sistema a um estado funcional. Técnicas de recuperação podem incluir reiniciar componentes falhos, restaurar dados de *backups*, reconfigurar o sistema ou migrar processos para outros nós saudáveis. O objetivo é trazer o sistema de volta a um estado consistente e confiável, minimizando o impacto da falha nas operações do sistema.

Neste texto, daremos atenção especial à detecção de defeitos ou erros (ver Figura 2.1), à comunicação em grupo e à replicação resiliente a falhas bizantinas. A comunicação em grupo, em particular, permite a entrega confiável de mensagens entre os membros de um sistema distribuído. Ao empregar protocolos como *multicast* confiável ou algoritmos de consenso, a comunicação em grupo garante que as mensagens cheguem a todos os destinatários pretendidos, mesmo na presença de falhas ou interrupções de rede. Essa confiabilidade ajuda a manter a consistência e a correção do sistema, o que é vital para a tolerância a falhas.

Entretanto, para cumprirem seus objetivos, tais mecanismos demandam custos computacionais adicionais, em termos de processamento e comunicação (Jalote 1994). Assim, para não comprometerem o funcionamento normal dos sistemas, os mecanismos de tolerância a falhas devem ser adequadamente configurados, de modo a conciliar esses custos adicionais com a expectativa de carga das aplicações, com os requisitos de desempenho especificados e com o grau desejado de confiabilidade [Dai e Wang 1992; Cristian 1991; Veríssimo e Rodrigues 2000].

As atividades de gerenciamento devem ser realizadas para assegurar a eficiência e eficácia destes mecanismos [Hegering et al. 1998]. Estas atividades consistem no monitoramento contínuo dos sistemas e dos mecanismos de tolerância a falhas e, também, na

realização de ajustes ou reconfiguração de parâmetros, quando as condições do ambiente ou os requisitos definidos pelas aplicações divergem daqueles considerados durante o projeto.

Neste contexto, um dos grandes desafios para a implementação e gestão dos mecanismos de tolerância a falhas é suportar as nuances dos ambientes distribuídos abertos (e.g., Internet de Coisas), os quais são constituídos por inúmeros dispositivos heterogêneos, conectados a partir de canais de comunicação com diferentes capacidades e sujeitos a condições de carga variadas. A consequência disto são ambientes caracterizados por incertezas em relação aos tempos de processamento e transmissão das mensagens.

Para lidar com essas incertezas, foram realizados diversos esforços de pesquisa relacionados, dentre outras coisas, à proposição de modelos de sincronia e mecanismos de tolerância a falhas adequados. As pesquisas relacionadas aos modelos de sincronia se preocupavam com a prova de correção dos algoritmos e protocolos face às incertezas temporais dos ambientes considerados -- ver, por exemplo, [Dolev et al. 1987; Lamport e Lynch 1989, Chandra e Toueg 1996; Cristian e Fetzer 1999; Veríssimo e Rodrigues 2000; Veríssimo e Casimiro 2002], entre outros. Em paralelo, muitas pesquisas relacionadas aos mecanismos de tolerância a falhas focaram na implementação de facilidades de adaptação -- ver, por exemplo, [Chockler et al. 1998; Macêdo 2000; Mills et al. 2004; Sivasubramanian et al. 2005; de Sá e Macêdo 2006; Karmakar e Gupta 2007; Macêdo 2008a], entre outros.

Para oferecer um desempenho adequado, estas facilidades de adaptação permitem aos mecanismos de tolerância a falhas ajustar dinamicamente seus parâmetros de configuração ver, por exemplo [Chen et al. 2002; Bertier et al. 2002; Falai e Bondavalli 2005; Rütti et al. 2006; Satzger et al. 2007], entre outros.

2.3.2. Aspectos Básicos do Projeto de Sistemas Físico-Digitais Distribuídos

Na medida em que novas facilidades de processamento e comunicação propiciam o surgimento de ambientes distribuídos físico-digitais com características dinâmicas, novos desafios se apresentam ao projeto e gerenciamento dos mecanismos de tolerância a falhas. Esta dinamicidade é originada de aplicações com características e requisitos dinâmicos, além de plataformas que permitem não apenas a alocação, liberação e migração dinâmica de recursos virtualizados, mas também a definição de composições dinâmicas, em que os componentes são definidos em tempo de execução [Macêdo 2008b] . Casos típicos desses ambientes distribuídos são encontrados, por exemplo, em plataformas de P2P (Peer-to-Peer), de grades computacionais e em ambientes de computação em nuvens [Milojicic et al. 2002; Baker et al. 2002; Rimal et al. 2009].

A dinamicidade acima referida dificulta o projeto e a gestão da maioria dos mecanismos de tolerância a falhas, que geralmente utilizam estimativas a respeito dos recursos e dos requisitos e características das aplicações para definirem adequadamente seus parâmetros operacionais. Os mecanismos adaptativos de tolerância a falhas existentes na literatura não estão, em geral, preparados para lidar com esta dinamicidade. Isto porque não consideram, na adaptação de seus parâmetros, a dinâmica do ambiente e dos requisitos das aplicações. Mais ainda, muitas das suposições usadas por estes mecanismos adaptativos, sobre o comportamento do ambiente, não são apropriadas. Em muitos ambientes, por exemplo, tais comportamentos são difíceis de serem caracterizados a partir de distribuições de probabilidades específicas.

Em sistemas distribuídos físico-digitais, mecanismos de tolerância a falhas devem ser adaptativos autonômicos (ver seção 2.3.3). Isto é, devem suportar a confiabilidade, adaptando

dinamicamente seus parâmetros de configuração a partir das mudanças observadas nas características dos ambientes distribuídos e nos requisitos de qualidade de serviço das aplicações [de Sá 2011]. Para isso, tais mecanismos autonômicos de tolerância a falhas devem considerar os seguintes requisitos de projeto [de Sá 2011]:

- Autogestão: atender aos desafios impostos pelos ambientes distribuídos modernos, a partir da proposição de mecanismos de tolerância a falhas com habilidade de autogerenciamento;
- Suporte aos requisitos dinâmicos de qualidade de serviço: apresentar e avaliar estratégias para implementação desses mecanismos, considerando o atendimento de requisitos dinamicamente definidos pelas aplicações ou usuários;
- Suporte ao legado de soluções existentes: permitir que os mecanismos propostos sejam implementados aproveitando as facilidades dos mecanismos de tolerância a falhas existentes, sem comprometer a correção dos mesmos;
- Reuso: conceber estratégias com baixo acoplamento em relação ao ambiente de execução, podendo atuar em diferentes modelos de sistemas distribuídos;
- Modelagem do comportamento dinâmico: explorar técnicas existentes na literatura que permitam a análise e síntese de modelos para lidar com o comportamento dinâmico do ambiente distribuído, sem comprometer o reuso da solução.

2.3.3. Visão Conceitual do Projeto de Mecanismos Autonômicos de Tolerância a Falhas

A ideia central no projeto é levar o problema da configuração dos parâmetros dos mecanismos de tolerância a falhas para um nível de abstração mais próximo das aplicações, seguindo a abordagem da computação autonômica [Horn 2001]. Isso implica no estabelecimento de novo modelo de alocação de responsabilidades no qual aplicações ou os usuários devem definir, em tempo de execução, às suas demandas ou restrições em termos de requisitos de qualidade de serviço (e.g. tempo de resposta, confiabilidade, percentual de uso de recursos etc.). Os mecanismos autonômicos de tolerância a falhas, por sua vez, devem observar as variações dinâmicas nas características do ambiente e ajustar seus parâmetros operacionais de modo a atender os requisitos dinamicamente definidos.

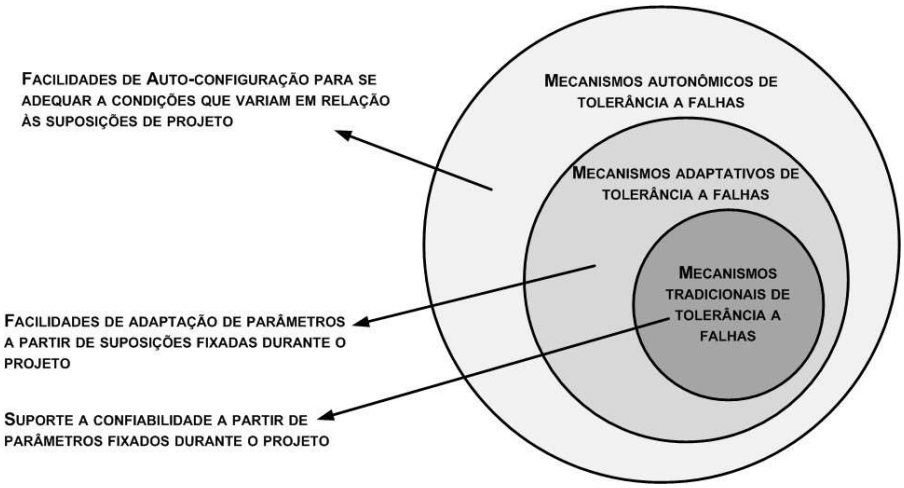

Figura 2.4. Visão conceitual dos mecanismos autonômicos de tolerância a falhas [de Sá 2011].

Conceitualmente, os mecanismos autonômicos para tolerância a falhas podem ser vistos como uma estratégia que provê o suporte à confiabilidade, encapsulando os mecanismos tradicionais ou adaptativos de tolerância a falhas (ver Figura 2.4):

- mecanismos tradicionais de tolerância a falhas se preocupam com o suporte à confiabilidade, garantindo as propriedades de correção de seu serviço, a partir de parâmetros de configuração fixados durante o projeto;
- mecanismos adaptativos de tolerância a falhas encapsulam (ou especializam) as facilidades dos mecanismos tradicionais para permitir a adaptação da configuração, considerando suposições (ou hipóteses) fixadas durante o projeto – e.g., distribuições de probabilidade, arranjo dos componentes, disponibilidade dos recursos etc;
- mecanismos autonômicos de tolerância a falhas encapsulam (ou especializam) as facilidades providas pelos mecanismos adaptativos, de modo a permitir a autoconfiguração quando as características do ambiente ou requisitos das aplicações variam em relação às suposições previamente definidas durante o projeto.

A implementação dos mecanismos autonômicos de tolerância a falhas não é uma tarefa simples, pois requer, por exemplo: modelos adequados para a representação do comportamento dinâmico do ambiente distribuído; a conciliação dinâmica de relações de compromissos associados aos requisitos de qualidade de serviços definidos etc.

2.3.4. Visão autonômica da gestão de tolerância a falhas

A concepção da habilidade de autogerenciamento (i.e., autoconfiguração) requer que os mecanismos de tolerância a falhas implementem um laço de gerenciamento autonômico (Horn, 2001), envolvendo monitoramento, planejamento e intervenções – ver Figura 2.5.

Figura 2.5. Laço de gestão implementado pelos mecanismos autonômicos de tolerância a falhas [de Sá 2011].

O monitoramento envolve a coleta de informações associadas às características dinâmicas do ambiente e à qualidade de serviço entregue pelos mecanismos de tolerância a

falhas. O planejamento envolve a definição de parâmetros operacionais adequados para levar o desempenho do mecanismo de tolerância a falhas a atender os requisitos de qualidade de serviço definidos, considerando o estado atual do ambiente distribuído. As intervenções envolvem a implantação, nos mecanismos de tolerância a falhas, dos parâmetros operacionais definidos durante o planejamento.

Este laço de gestão se assemelha aos laços de controle implementados no campo dos sistemas industriais [Ogata 1995].

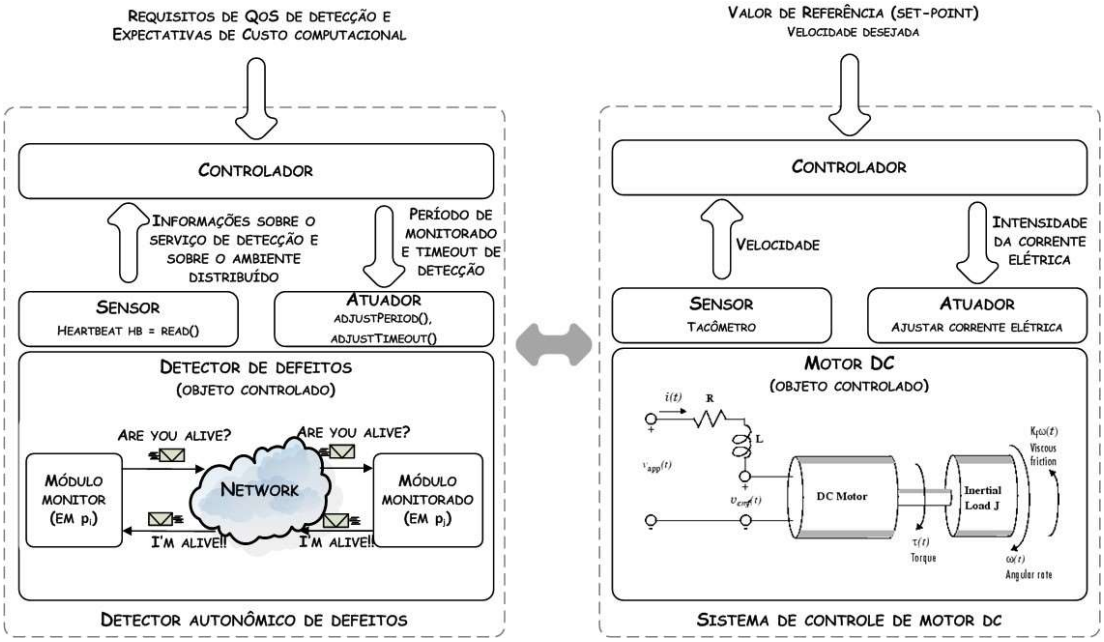

Figura 2.6. Comparativo entre um mecanismo autonômico de detecção de defeitos e um sistema de controle de motor [de Sá 2011].

Como exemplo, pode-se fazer o paralelo entre um mecanismo autonômico de detecção de defeitos e um sistema de controle de motor DC (corrente contínua), ver Figura 2.6.

Nesse exemplo, um tacômetro (sensor) monitora a velocidade do motor (objeto controlado). Em seguida, um controlador verifica se a velocidade observada está de acordo com um valor de referência (*set-point*). Existindo desvios entre as velocidades, se determina um novo valor de corrente elétrica (configuração) para o motor DC, de modo que o mesmo apresente a velocidade desejada. Então, um atuador ajusta a corrente elétrica do motor, usando o valor informado pelo controlador. No caso do detector de defeitos, métodos de leitura (sensor) coletam informações sobre comportamento do ambiente e o desempenho do serviço de detecção (objeto controlado), usando mensagens de monitoramento, por exemplo. Em seguida, o controlador verifica se o desempenho observado está consoante com os requisitos de qualidade de serviço de detecção e as expectativas de custo definidas pelo usuário (i.e., *set-points*). Existindo divergências, o controlador determina os novos parâmetros do detector (e.g. período de monitoramento de falhas e *timeout* de detecção), de modo que o detector apresente o desempenho desejado em termos de qualidade de serviço de detecção e custo computacional. Em seguida, o controlador ativa métodos de escrita (atuador) necessários para que os novos parâmetros de configuração sejam implantados.

Assim, na perspectiva do projeto, o laço de gestão dos mecanismos autonômicos de tolerância a falhas pode ser projetado seguindo a mesma abordagem usada nos sistemas de

controle, isto é: embutindo, nos mecanismos de tolerância a falhas existentes, controladores (ou gestores autonômicos) responsáveis pelo planejamento das configurações, além de métodos de sensoriamento e atuação que realizam a coleta das informações e a implantação dos novos parâmetros de configuração, respectivamente.

Por conta das técnicas de análise e síntese disponibilizadas pela teoria de controle de sistemas dinâmicos [Ogata 1995; Hellerstein et al. 2004], a implementação dos mecanismos autonômicos de tolerância a falhas pode utilizar o mesmo ferramental matemático para a modelagem de suas funções autonômicas.

2.4. Estudos de Caso

Esta seção apresenta estudos de caso nos quais a habilidade de gestão autonômica é implementada em mecanismos e protocolos básicos para a construção de sistemas distribuídos. Nestes estudos de casos, aspectos de implementação da gestão autonômica, como identificação de políticas de alto nível, variáveis a serem monitoradas e parâmetros controláveis em possíveis laços de controle, são apresentados, incluindo o mapeamento entre os requisitos de alto nível e os laços de controle. Também são apresentadas algumas estratégias para avaliação do desempenho destes mecanismos e protocolos autonômicos.

Existe uma quantidade considerável de conhecimento prático e experiência na aplicação de paradigmas e ferramentas em diversas situações e ambientes de execução para projetar e programar sistemas ciberfísicos complexos de forma segura e confiável. Essa *expertise* é ilustrada por diversos exemplos prototípicos e estudos de caso. Nesta seção, apresentaremos brevemente alguns desses estudos de caso e provas de conceito, listados abaixo, a fim de auxiliar os leitores na compreensão dos principais desafios e abordagens de soluções existentes.

2.4.1. Detectores de defeitos

Neste estudo de caso, é apresentado como detectores de defeitos para sistemas distribuídos podem ser especializados para implementar a habilidade de gestão autonômica. Por questões didáticas e de maior simplicidade em termos das propriedades e funcionamento dos detectores de defeitos, este estudo de caso será discutido sem detalhar equações utilizadas nos algoritmos. Uma descrição mais detalhada pode ser encontrada em [de Sá e Macêdo 2010; de Sá 2011]. Neste texto, discutiremos como permitir, em tempo de execução, que os detectores de defeitos autoadaptem seus parâmetros operacionais (i.e., período de monitoramento e *timeout* de detecção de defeitos), considerando métricas de qualidade de serviço de detecção e mudanças nas características do ambiente distribuído.

Detectores de defeitos são elementos básicos para garantia da confiabilidade em sistemas distribuídos, pois, na ocorrência de falhas, permitem a ativação de procedimentos de recuperação ou de reconfiguração do sistema. Em sistemas distribuídos, a detecção de defeitos é realizada a partir da troca de mensagens de monitoramento entre processos do sistema.

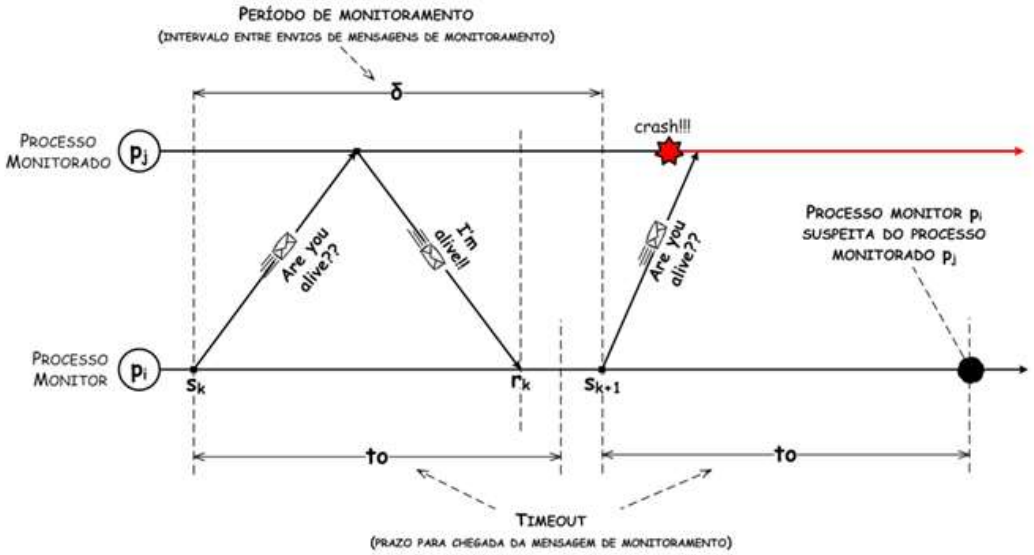

Figura 2.7. - Modelo Pull de Monitoramento de Defeitos

Dentre os modelos de monitoramento existentes, no modelo Pull [Felber 1998] (Figura 2.7), um processo monitor periodicamente envia uma mensagem de monitoramento *"are you alive?"* para o processo monitorado, que deve responder com uma mensagem *"I'm alive!"* – ver Figura 2.7. A cada *período de monitoramento*, caso o processo monitor não receba a mensagem de *"I'm alive!"* dentro de uma determinada janela de tempo (também chamada de *timeout de detecção*), então ele suspeita da falha do processo monitorado.

Para exemplificar a implementação de um detector autonômico de defeitos, este estudo de caso considera a arquitetura proposta em [de Sá e Macêdo, 2010] – ver Figura 2.8. Esse detector de defeitos oferece um nível de autonomia classificado como **Adaptativo Autonômico**.

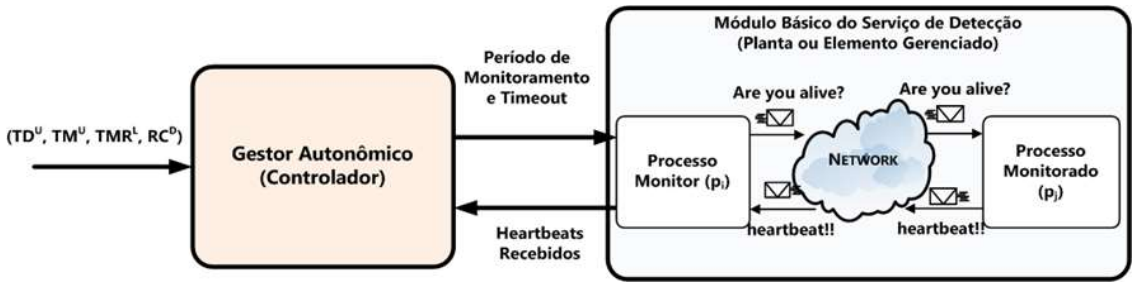

Figura 2.8. Arquitetura do Detector Autonômico de Defeitos [de Sá e Macêdo 2010]

A arquitetura do detector em [de Sá e Macêdo 2010] considera a implementação de um gestor autonômico responsável por realizar, em tempo de execução, a adaptação de parâmetros do detector de defeitos, considerando: os requisitos dinâmicos de aplicação definidos em termos de um Acordo de Nível de Serviço (SLA, Service Level Agreement) de detecção de defeitos, no qual se especifica a velocidade mínima, a confiabilidade mínima e o custo máximo da detecção de defeitos; e a qualidade de serviço (QoS) de detecção de defeitos, na qual se estima a velocidade, a confiabilidade e o custo da detecção.

A QoS de detecção de defeitos depende da definição adequada do *timeout de detecção* e do *período de monitoramento*. Timeouts de detecção muito longos podem tornar a detecção lenta. Por outro lado, timeouts muitos curtos podem levar o detector a apontar falsas suspeitas

de falhas no sistema, como nos casos em que há oscilações nos tempos de processamento ou de comunicação, por exemplo. Períodos de monitoramento muito curtos podem consumir muitos recursos computacionais e provocar ou ampliar congestionamentos nos canais de comunicação entre os processos do sistema. Períodos de monitoramento muito longos podem tornar a detecção muito lenta. Além disso, no caso de perda de mensagens, por exemplo, períodos de monitoramento longos tornam maiores os intervalos de tempos que o detector leva para corrigir falsas suspeitas de falha.

Para atender às relações de compromisso entre velocidade, confiabilidade e custo da detecção de defeitos, o detector autonômico em [de Sá e Macêdo 2010] implementa dois laços de ajuste de parâmetros. O primeiro laço ajusta dinamicamente o timeout de detecção, considerando o desvio entre a confiabilidade mínima e a observada: se a confiabilidade de detecção é superior à mínima, o timeout de detecção é decrementado para aumentar a velocidade de detecção; caso contrário, se a confiabilidade de detecção está abaixo da mínima, o timeout de detecção é incrementado para aumentar a confiabilidade de detecção. O segundo laço ajusta dinamicamente o período de monitoramento, considerando o desvio entre o custo de detecção máximo e o custo de detecção estimado: se o custo de detecção é inferior ao mínimo, então o período de monitoramento é decrementado para aumentar a velocidade de detecção; caso contrário, o período de monitoramento é incrementado para reduzir o custo de detecção. Ambos os laços de adaptação de parâmetros implementados no detector autonômico em [de Sá e Macêdo 2010] restringem os incrementos realizados no período de monitoramento e no timeout de detecção, observando a velocidade de detecção mínima esperada.

Nas abordagens do tipo **adaptativo autonômico** de detecção de defeitos, existem dois desafios significativos na implementação dos laços de adaptação de parâmetros: (a) estimar em tempo de execução a qualidade de serviço do detector de defeitos; e (b) estimar o tamanho do ajuste a ser realizado nos parâmetros do detector de defeitos de modo que o efeito desejado se reflita no desempenho do detector de defeitos.

No caso do detector autonômico em [de Sá e Macêdo 2010], a estimativa da QoS de detecção de defeitos considera a seguinte abordagem:

- **Velocidade de detecção** – obtida a partir da estimativa online do tempo de detecção, a qual é realizada, no início de um ciclo de monitoramento, baseado no atraso de interação entre os processos monitor e monitorado e em uma expectativa de pior caso para expiração do timeout de detecção.
- **Confiabilidade da detecção** – estimada a partir de uma análise do número, do intervalo entre ocorrências e da duração das falsas suspeitas de falhas.
- **Custo da detecção** – representa o percentual do uso dos recursos disponíveis e é estimado a partir do atraso na interação entre os processos do sistema, considerando que atrasos de interação maiores refletem potenciais reduções na quantidade de recursos disponíveis.

Na abordagem apresentada em [de Sá e Macêdo 2010], o atraso de interação entre os processos do sistema distribuído diz respeito ao intervalo de tempo necessário para que um processo do sistema receba uma nova mensagem (i.e., informação atualizada) de outro processo do sistema. Esse atraso é influenciado, não apenas pelos períodos de monitoramento, mas também pelos atrasos fim-a-fim e perdas de mensagens nos canais de comunicação, isto é: quanto maiores os atrasos e as perda de mensagens nos canais de comunicação, maiores são

os atrasos na interação entre os processos do sistema; da mesma forma quanto maior for o período de monitoramento, maior também será o atraso na interação entre processos monitor e monitorado.

Por fim, em [de Sá e Macêdo 2010], a estimativa do tamanho do ajuste a ser realizada nos parâmetros do detector autonômico de defeitos é realizada usando estratégias da teoria de controle realimentado [Ogata 1995]. Na abordagem o ajuste realizado é proporcional ao tamanho do desvio entre o valor desejado em termos de SLA e o valor atual em termos de QoS de detecção. O valor do peso dado ao ajuste nos parâmetros é estimado usando estratégias da teoria de controle, mas como o sistema distribuído é dinâmico e com respostas não lineares, esses pesos também são adaptados continuamente quando são percebidas mudanças no comportamento do sistema em termos dos atrasos de interação – ver maiores detalhes em [de Sá e Macêdo 2010].

2.4.2. Comunicação em Grupo

Neste estudo de caso, é apresentado como protocolos de comunicação em grupo podem ser especializados para apresentar a habilidade de gestão autonômica, permitindo que tais protocolos ajustem seus parâmetros de configuração de acordo com a carga de trabalho do ambiente computacional e de modo a se adequar ao nível de consumo de recursos desejado por uma aplicação ou pelo usuário. Este estudo de caso será apresentado principalmente para discutir como a implementação dos loops autonômicos pode ser realizada sem comprometer propriedades de correção do protocolo de comunicação em grupo.

Protocolos de comunicação em grupo se apresentam como uma poderosa abstração para o desenho de aplicações distribuídas tolerantes a falhas em uma variedade de cenários que expressam diferentes modelos de sistemas distribuídos, de sistemas síncronos a sistemas assíncronos. Embora similar em princípios, diferentes especificações e implementações foram propostas para os diferentes modelos de sistema [Birman 1993; Macêdo et al. 1993; Ezhilchelvan et al. 1995; Cristian 1996; Chandra 1996; Chockler et al. 2001; Défago et al. 2004]. De forma resumida, um protocolo de comunicação em grupo provê o serviço de entrega do mesmo conjunto de mensagens a um mesmo grupo de processos que cooperam em uma computação distribuída. Desta forma, um protocolo de comunicação em grupo abstrai, para camadas de aplicação superiores, a complexidade e incerteza do sistema de comunicação subjacente, como falhas ou entrega de mensagens fora de ordem, permitindo, por exemplo, o provimento de replicação ativa de servidores.

O protocolo de comunicação em grupo, por sua vez, utiliza um conjunto de algoritmos básicos, como difusão confiável e consenso. Por exemplo, o problema de determinar a visão deste grupo, ou seja, o conjunto de processos que o compõe, conhecido como *membership*, pode utilizar como componente da solução o consenso distribuído, ou seja, os processos concordam em um mesmo e único conjunto de processos que formam a visão atual deste grupo. Ainda, o problema da entrega ao grupo em uma mesma ordem de um mesmo conjunto de mensagens, dito difusão atômica confiável, é equivalente ao consenso distribuído, uma vez que os processos envolvidos executam uma forma de acordo [Chandra et al. 1996].

Conforme o modelo de sistema que caracteriza o cenário de atuação do protocolo de comunicação em grupo, pode-se determinar a qualidade de serviço que pode ser obtida, em termos de garantias na entrega das mensagens e na visão dos membros do grupo. O protocolo apresentado de [Macêdo e Freitas 2009] usa uma matriz de registro de relógio vetorial denominado *Timed Causal Blocks*, estrutura introduzida em [Macêdo 2007], que em ambiente

síncrono utiliza-se de limites temporais para terminação do protocolo. Por exemplo, na Figura 2.9, ilustra a matriz de blocos causais (*Causal Block Matrix*) em ambiente assíncrono, conforme introduzido em [Macêdo et al. 1993], para um grupo de 6 processos, onde o bloco 2 está completo em face dos processos p_2, p_5, e p_6 haverem enviado mensagens com número 2, e os processos p_1, p_3, e p_4, enviado mensagens com número 3. A completude do bloco 2 permite a entrega das mensagens de número 2 de forma determinística em todos os processos.

	p_1	p_2	p_3	p_4	p_5	p_6
1	+			+		
2		+			+	+
3	+		+	+		
4	+				+	
5		+				

Figura 2.9. Exemplo de Matriz de Blocos para grupo de 6 processos

Em um ambiente síncrono, usando o modelo de *Timed Causal Blocks,* é possível calcular limites temporais para a completude de blocos, onde após a criação de um bloco um processo que ainda não se manifestou em prol desta completude se manifesta em um tempo limite denominado *time silence* (*ts*). Dessa forma, um bloco causal b se torna completo tão logo uma das duas seguintes condições se verifique: (i) bloco b esteja temporalmente completo ou bloco b esteja completo logicamente (usando tempo lógico). Com o monitoramento adequado do nível de qualidade de serviço dos canais de comunicação, o mecanismo base adapta-se a esta informação e pode prover o melhor ajuste, ou seja, se houver garantias temporais, otimizar o desempenho, e, caso contrário, utilizar tais rótulos temporais apenas para suspeita de falhas por parada. Mais detalhes sobre essa abordagem podem ser encontrados em [Macêdo 2007; Macêdo e Freitas 2009; Macêdo e Freitas 2010].

Neste texto, exploraremos cenários dinâmicos de sistemas físico-digitais distribuídos, como apresentados nas seções anteriores. Neste contexto, protocolos de comunicação em grupo devem se auto ajustarem de acordo com mudanças na carga de trabalho ou nos requisitos da aplicação. Este ajuste deve ser realizado por meio da configuração dinâmica dos parâmetros de operação. Por exemplo, suponha um ambiente de computação em nuvem onde um módulo de gerenciamento de níveis de serviço denominado *SLA Manager* (de *Service Level Agreement*) define o nível de serviço negociado para a aplicação distribuída que executa na nuvem. Como uma ilustração, considere o esboço de um gerenciador de recursos da nuvem, *Resource Cloud Manager*, o qual conforme o *SLA* atual, requisita mudanças nos pontos de operação (*setpoint*) para o consumo de recursos dos protocolos subjacentes, como, por exemplo, de um protocolo de comunicação em grupo, o qual é utilizado em todas as réplicas em execução. Uma mudança neste *setpoint* pode implicar no aumento de consumo de recursos, provendo entrega de mensagens mais rápida, ou, ao contrário, reduzir o consumo de recursos para as aplicações associadas, implicando em maior retardo na entrega das mensagens. Para lidar com esses cenários, equipamos o protocolo base de comunicação em grupo para ser autogerenciável, percebendo o nível de serviço requerido para o usuário, em

termos de consumo de recursos, e ajustando os seus parâmetros operacionais de forma dinâmica.

Assumimos um modelo de sistema distribuído parcialmente síncrono, no qual o sistema torna-se síncrono em um tempo arbitrário [Dolev et al. 1987]. Sob estas premissas de modelo, construímos um protocolo de comunicação em grupo básico que assume um mecanismo subjacente que permite inferir qualidade de serviço dos processos do grupo. Desta forma, utilizamos uma matriz de blocos que registra o envio/recepção de uma mensagem e define *timeouts* de completude dos blocos e outras condições de entrega de mensagens.

A proposta autogerenciável estende o protocolo de comunicação em grupo base, de modo a tratar da escolha apropriada do parâmetro de configuração do protocolo, no caso, o tempo máximo sem transmitir mensagens em um grupo, denominado de *time-silence*. Escolher o valor apropriado do *time-silence* implica em uma relação de compromisso: a) períodos longos para *time-silence* diminuem o *overhead* do protocolo e o consumo de recursos, contudo, podem determinar um tempo de bloqueio maior na entrega de mensagens, pois caso um processo não envie mensagem de aplicação para completude dos blocos, este só contribui para a completude após o período dado pelo *time-silence*; b) do outro lado, períodos curtos de *time-silence* podem reduzir o tempo de bloqueio, porém ao custo do aumento do *overhead* e do consumo de recursos, o que pode ser problemático quando o ambiente estiver sujeito a alta carga de trabalho, implicando em aumento do atraso fim-a-fim e tempo de bloqueio na entrega de mensagens.

Utilizamos elementos da teoria de controle realimentado [Ogata 1995; Hellerstein et al. 2004] para o ajuste de parâmetros de protocolos de computação distribuída, de acordo com um dado nível de serviço. A ideia básica do controle realimentado é que um controlador utilize de sensores que observam as saídas atuais de um objeto controlado (ou planta), e executa um algoritmo de controle (ou lei de controle) para definir novos valores de entrada de modo que as saídas sigam o valor de referência (*setpoint*) para o comportamento desejado. Neste caso, a planta controlada é o protocolo de comunicação baseado no mecanismo de *timed causal blocks* [Macêdo 2007]. O sensor coleta e pré-processa dados sobre o ambiente computacional e desempenho do protocolo de comunicação em grupo, como, por exemplo, latência fim-a-fim dos canais de comunicação, *overhead* e carga de trabalho. O controlador utiliza desta informação para ajustar de forma dinâmica o *time-silence* conforme o valor de referência. Este valor de referência é definido em termos do nível de consumo de recursos contratado pela aplicação. A Figura 2.10 apresenta uma visão simplificada do laço de controle.

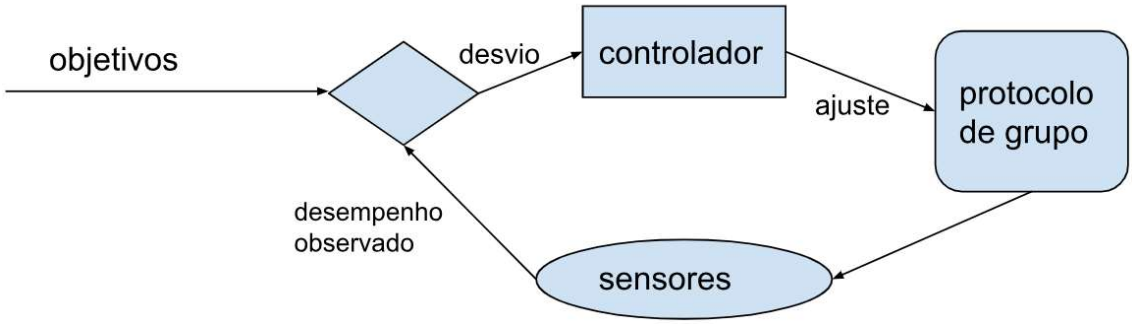

Figura 2.10. Laço de controle para auto configuração do protocolo de comunicação em grupo

Estimamos o consumo de recursos por meio de uma relação que associa as latências do ambiente aos recursos disponíveis em um dado momento: a maior disponibilidade de recursos (como largura de banda, memória, processamento etc.) produz menores latências de comunicação e processamento, e vice-versa.

A partir destes valores observados, e de uma estimativa da latência de comunicação atual do sistema (uma vez que o sistema não é síncrono), construímos a metáfora do consumo de recursos do sistema, utilizando uma relação linear entre a variação da latência observada, dita *jitter* e a quantidade de recursos consumidos do ambiente: se o consumo maior de recursos é alto, há poucos recursos de comunicação e processamento disponíveis, observando valores de latências maiores; já se há pouco consumo, esta maior disponibilidade de recursos reflete em latências observadas baixa.

É importante ressaltar que a relação entre consumo de recursos e latência possivelmente não apresenta linearidade. O escopo deste trabalho não inclui a identificação do sistema de modo a estimar a relação mais adequada, de modo que experimentamos a relação linear para avaliar a eficiência do algoritmo proposto a partir desta premissa.

As condições do ambiente dependem da carga de trabalho do sistema e do tamanho do grupo (número de nós membros). Uma vez que estas condições podem variar em tempo de execução, pode-se dinamicamente calcular referências que expressem: a) requisitos do usuário em termos de consumo de recursos desejado; b) consumo atual de recursos do ambiente; e c) número de membros ativos do grupo.

Assumimos que o *overhead* do protocolo influi no consumo de recursos, de forma proporcional: quanto maior o *overhead* do protocolo, maior o número de mensagens enviadas pelo protocolo, e assim maior o percentual de recursos consumidos. Ou seja, para o protocolo de exemplo quando todos os processos continuamente enviam mensagens ao grupo, não há atuação do mecanismo de *time-silence* e nenhuma mensagem de controle (*overhead* do protocolo); por outro lado, quando apenas um membro do grupo está ativo, os demais $n - 1$ membros contribuem sempre para a completude por meio do *time-silence* – ou seja, situação de maior *overhead* do protocolo.

Neste caso específico, quanto maior o período de *time-silence* é menor a frequência de atuação deste mecanismo e é menor o *overhead* devido às mensagens de controle do protocolo. Assumindo linearidade nesta relação e a relação linear entre *overhead* e consumo de recursos, podemos associar a um dado consumo de recursos desejado um valor de *overhead* e, por meio de uma lei de controle, o ajuste necessário no período de atuação do *time-silence*.

Para isto, temos um componente sensor que percebe as latências atuais da rede e converte estas métricas mensuradas (transdução), estimando: i) o valor atual e máximo do *overhead* do protocolo; ii) os valores máximo, mínimo e a média das latências de comunicação fim-a-fim, de forma adaptativa de modo a obter amostras mais significativas da dinâmica atual; iii) o consumo de recursos atual; e iv) carga de trabalho.

O controlador mapeia o percentual de recursos disponíveis a serem alocados (diferença entre a quantidade de recursos desejada e a quantidade de recursos consumidos) em um *setpoint* dinâmico do *overhead* do protocolo. A escolha da quantidade de recursos desejada é uma decisão da aplicação usuária, conforme o cenário de execução do protocolo. Uma lei de controle proporcional define o período do *timesilence* em função da diferença entre o *overhead* atual e o desejado: se o valor de *overhead* atual é menor que o desejado, então

reduz-se o período com mais mensagens de controle, consumindo mais recursos da rede e diminuindo o tempo de bloqueio; caso contrário, se o valor atual de *overhead* é maior que o desejado, então o período é incrementado, diminuindo as mensagens de controle e o consumo de recursos da rede e aumentando o tempo de bloqueio.

Em comparação com o protocolo base, observamos em experimentos melhores resultados do protocolo de grupo autogerenciável. Mesmo em cenários dinâmicos e sob falhas, a versão autogerenciável mantém o ponto de operação para o *overhead* desejado, inclusive na mudança de carga para diferentes tamanhos de grupo. Detalhes da implementação desta abordagem e da avaliação de desempenho comparativa com o protocolo base podem ser encontrados em [Macêdo et al. 2013].

2.4.3. Replicação Bizantina

Neste estudo de caso, discutimos a implementação da habilidade de autogestão em um protocolo clássico de replicação tolerante a falhas bizantinas, o PBFT [Castro e Liskov 2002], permitindo que o mesmo se ajuste automaticamente de acordo com o monitoramento do ambiente ao longo da execução. Este estudo de caso também será discutido de forma simplificada e tem o papel principal de exemplificar a implementação de um mecanismo semi-autonômico que realiza adaptação online, mas não permite que as políticas de adaptação sejam mudadas em tempo de execução.

Replicação é um mecanismo utilizado para elevar o nível de serviço de serviços computacionais distribuídos, baseados no paradigma cliente/servidor, face à possibilidade de falhas do servidor. Uma técnica utilizada para replicação é a máquina de estados replicada [Schneider 1990] em que cada uma das N máquinas que replicam o serviço implementam uma máquina de estados completa do serviço computacional, ou seja, o conjunto de estados internos que caracterizam o serviço provido e as transições realizadas de um estado a outro na ocorrência de uma dada operação. De modo a reproduzir o mesmo comportamento, todos os servidores replicados iniciam em um mesmo estado e as máquinas de estados são determinísticas, ou seja: em face de uma dada operação em um dado estado, sempre executam a mesma transição de estados. O conjunto de servidores deve concordar, ou chegar ao consenso, em relação à ordem de execução das requisições dos clientes, e baseado nestas premissas, todos servidores devem executar o mesmo conjunto de operações e apresentar o mesmo conjunto de resultados.

Cenários sujeitos somente a falhas por parada são menos desafiadores, mas cenários em que as máquinas podem falhar por comportamento arbitrário, devido a falhas latentes ou transientes, ou mesmo por intrusão de forma maliciosa, requerem uma abordagem mais cuidadosa. Em 1982, o clássico problema dos Generais Bizantinos [Lamport et al. 1982] apresentou as condições para o consenso distribuído diante de falhas arbitrárias, a partir de então denominadas falhas bizantinas. Neste mesmo artigo foi provado que o consenso bizantino precisa de $N = 3f + 1$ para tolerar o máximo de f processos falhos, em um sistema distribuído síncrono (limites superiores conhecidos para tempos de transmissão de mensagens e execução de passos computacionais).

A partir do trabalho de [Castro e Liskov 2002] sobre um protocolo de replicação tolerante a falhas bizantinas, dito PBFT (*Practical Byzantine Fault Tolerance*), que a replicação bizantina ganhou interesse, por propor um conjunto de otimizações, tais como técnicas de rejuvenescimento de réplicas para recuperação proativa, uso de *batching* para otimizar o processamento de conjunto de requisições, mudanças de visões periódicas para

rotacionar o papel do coordenador, uso de *checkpoint* periódico para *garbage collection*, em um cenário que é naturalmente custoso para implementação. Desta forma, há um conjunto de parâmetros relacionados às técnicas de otimização (como tamanho da janela de *batch*, período de *checkpoint*, período de ressincronização de estado), que devem ser ajustados de forma apropriada para garantir o desempenho desejado.

Conceber estratégias de adaptação de parâmetros para um protocolo de replicação bizantina não é uma tarefa simples, uma vez que a implementação destes protocolos é realizada considerando diferentes aspectos – como, por exemplo, segurança, gestão de memória, gerenciamento de réplicas, sincronização de estados. Estes diferentes aspectos, relacionados ao protocolo de replicação, tornam a análise extremamente complexa e dificultam a obtenção de estratégias adequadas de adaptação de parâmetros operacionais. Nesse sentido, a estratégia de ajuste dinâmico ora apresentada foi concebida abstraindo os detalhes internos de implementação. Para tanto, introduzimos aqui um modelo abstrato no qual os diferentes mecanismos, sub-protocolos e procedimentos do protocolo são vistos como caixas pretas organizadas de forma a compor um pipeline abstrato. Os estágios desse pipeline abstrato são selecionados considerando as etapas de interesse da adaptação de parâmetros. A partir do pipeline abstrato, propomos o aPBFT (*Adaptive PBFT*), uma extensão adaptativa do PBFT, a qual baseia a sua adaptação no ajuste dinâmico do tamanho da janela batch (*BS*, *Batch Size*) e do timeout de batching (*BT*, *Batch Timeout*). A escolha dos parâmetros de adaptação se deve ao fato do mecanismo de batching representar um ponto importante para conciliar a carga das aplicações com os custos computacionais associados ao fluxo de execução do protocolo de replicação.

O aPBFT realiza estimativas a respeito da carga imposta pelos clientes e sobre o desempenho do protocolo básico (PBFT). Para isto, a estratégia proposta implementa um laço de controle que usa informações obtidas a partir dos eventos associados ao envio, recebimento e processamento de mensagens do PBFT e determina os novos parâmetros do *batching*. As subseções a seguir apresentam em maiores detalhes os principais aspectos relacionados à implementação do aPBFT, como apresentado na Figura 2.11.

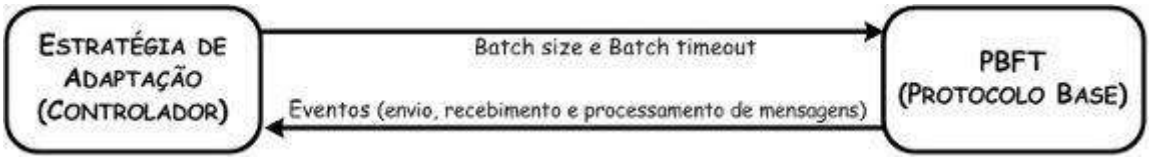

Figura 2.11. Laço de controle para a versão adaptativa do PBFT [de Sá et al. 2013]

O aPBFT abstrai o fluxo de atendimento de requisições do PBFT considerando um *pipeline* abstrato com quatro estágios, denominados: *checking*; *buffering and batching*; *ordering*; e *processing* (ver Figura 2.12).

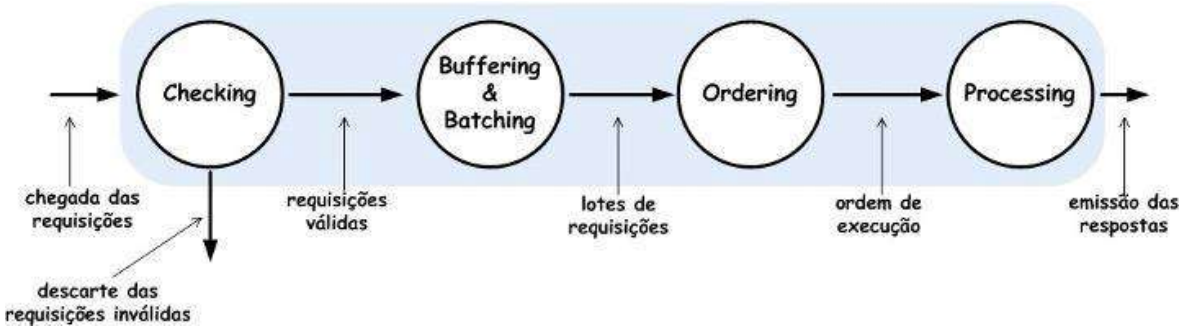

Figura 2.12. Pipeline abstrato representando o processamento de requisições no PBFT [de Sá et al. 2013]

O estágio de *checking* corresponde aos procedimentos realizados pelo PBFT para verificar a integridade, autenticidade e validade das requisições dos clientes. Requisições válidas são repassadas para o mecanismo de *batching*, enquanto que requisições inválidas são descartadas. No estágio de *buffering and batching*, as requisições aceitas são armazenadas em um *buffer*. Quando o número de requisições é suficiente para completar um lote (ou *batch*) de requisições, ou seja, o número de requisições é maior ou igual ao *batch size*, ou quando o *timeout* de *batch* expira, a réplica primária assinala este lote com um número seqüencial e, então, se inicia a fase de *ordering*. No estágio de *ordering*, as réplicas executam o acordo sobre a ordem das requisições. No estágio de *processing*, os lotes efetivados são processados e, então, cada réplica envia as respostas das requisições aos respectivos clientes solicitantes.

Evidentemente, esta descrição baseada no pipeline abstrai uma série de nuances do protocolo de replicação. Durante a fase de ordenação, por exemplo, é possível que várias instâncias do sub-protocolo de acordo sejam executadas em paralelo. Todavia, o uso do pipeline abstrato coloca o protocolo de replicação em um nível de detalhe adequado para a implementação da estratégia de adaptação. A estimativa do *timeout* de *batching* se baseia no intervalo médio de tempo para ordenação e execução das requisições, como apresentado na Figura 2.13. Neste texto, este intervalo de tempo é dito *MTE*, *Mean Time To Ordering and Execute*.

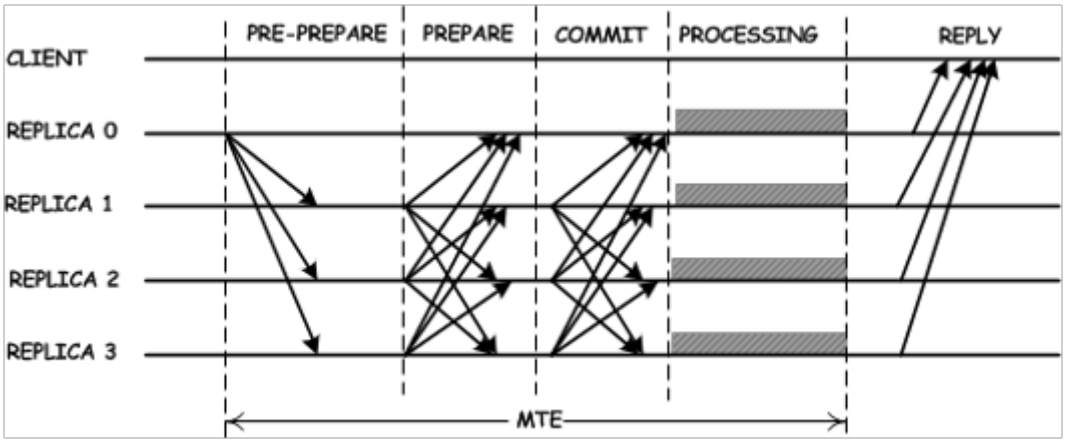

Figura 2.13. Tempo médio para ordenação e execução das requisições (MTE) no PBFT [de Sá et al. 2013].

A ideia central, usada pela estratégia de adaptação, é que o *timeout* de *batching* deve ser longo o suficiente para permitir o acúmulo de requisições para a composição do lote, mas, por outro lado, o mesmo deve ser curto o suficiente para permitir que pelo menos um lote de

requisições esteja no estágio de *ordering* – de modo a manter tal estágio sempre ocupado, na medida em que existem requisições a serem ordenadas e processadas. Entretanto, se o intervalo médio entre chegadas das requisições dos clientes (dito, *MTA – Mean Time Between Arrivals*) é maior que o *MTE*, então o *timeout* de *batching* pode ser zero, uma vez que a carga imposta pelos clientes é muito baixa e os estágios de ordering e processing do pipeline estão operando em um ritmo menor que suas capacidades de execução.

Estimamos o tamanho do *batch* a partir da relação entre o *MTE* e *MTA*. A ideia central para o ajuste é a mesma usada na adaptação do *timeout* de *batching*, isto é, *BS* deve ser definido de tal forma que permita ao menos um lote de requisições esteja nos estágios *ordering* ou *processing*. Assim, se os estágios *ordering* e *processing* levam em média *MTE* unidades de tempo para realizar em conjunto suas atividades, então o lote deve conter o número de requisições que foram possíveis de ser acumuladas neste intervalo de tempo.

O componente sensor percebe os eventos temporais relevantes no pipeline apresentado para realizar o cálculo dos intervalos de *MTE* e *MTA* ora citados. Já o componente controlador realiza os procedimentos de ajuste do *timeout* e do tamanho de *batch*. Para adaptação do *timeout* de *batching*, estima-se a carga das aplicações, e utiliza-se o valor do intervalo médio para ordenação e execução como *timeout* de *batching*. Deve-se observar que se a carga das aplicações é inferior à capacidade dos estágios de *ordering* e *processing*, logo o sistema está ocioso e o *timeout* de *batching* é definido com valor zero (não é realizado lote de requisições). Por fim, adaptamos o tamanho do *batch* usado pelo PBFT: o tamanho do *batch* reflete a relação entre a carga do sistema e o intervalo médio para ordenação e execução. Caso não possamos estimar ainda este intervalo médio por conta, por exemplo, de uma mudança de visão, o valor é definido como um (não é realizado lote de requisições).

Em comparação com o protocolo clássico PBFT, observamos em experimentos melhores resultados do aPBFT, em especial quando há mudanças na dinâmica do cenário (como carga variável), uma vez que o ajuste dinâmico de parâmetros amortiza os efeitos de alta carga de trabalho com maiores valores para o *timeout* e tamanho do *batch* e, ao mesmo tempo, o protocolo aPBFT se ajusta para rápido processamento dos lotes em condições de baixa carga – isto é, aPBFT trabalha como um mecanismo de controle de fluxo que auto-ajusta automaticamente o fluxo da máquina de estados. Detalhes da implementação desta abordagem e da avaliação de desempenho comparativa com o protocolo original podem ser encontrados em [de Sá et al. 2013].

2.4.4. Livro-razão distribuído

Um livro-razão distribuído, dito blockchain, permite o armazenamento confiável e seguro de registros de transação em um conjunto de processos conectados através de uma rede, sem requerem nenhuma entidade centralizada garantidora [Gamage et al. 2020]. Blockchains combinam computação distribuída e segura mantendo uma estrutura de dados dita cadeia de blocos que garante a persistência de transações armazenadas pelos processos que integram o sistema. As blockchains podem ser permissionadas (privadas) e não-permissionadas (públicas).

Uma blockchain permissionada requer que os processos sejam conhecidos a priori e autenticados para participarem do sistema. Um exemplo é a Hyperledger Fabric [Androulaki et al. 2018]. Por outro lado, nas blockchains não-permissionadas qualquer processo pode participar. Os processos não precisam sequer confiar uns nos outros. Exemplos incluem a Bitcoin [Nakamoto 2008] e Ethereum 1.0 [Wood 2014]. Em geral, há uma criptomoeda

associada a blockchain, sendo registrada na plataforma a movimentação destes ativos. Estas blockchains utilizam mecanismos de consenso não-determinísticos em geral, como a "Prova-de-Trabalho" (Proof-of-Work - PoW), a qual valida novos blocos com base na capacidade de processamento dos usuários, o que resulta em alto gasto energético.

Em blockchains permissionadas, há um controle prévio dos membros e o uso de algoritmos clássicos de consenso, como Raft [Ongaro e Ousterhout 2014] e PBFT [Castro e Liskov 2002]. Sistemas amplos não permissionados podem incluir uma quantidade expressiva de nós, mas apresentam uma vazão de transações bem limitada em comparação a sistemas de menor escala baseados em um grupo definido e algoritmos de consenso convencionais.

De maneira generalizada, podemos pensar em blockchain como uma forma de encapsular a máquina replicada de estados baseada em consenso, em que as transações são registradas em blocos e há uma estrutura de dados própria, a dita cadeia de blocos, a qual é resiliente à adulteração. Tal resiliência reside no fato de que cada bloco, desde o primeiro - denominado gênese – consiste em um conjunto de transações bem como do sumário criptográfico (hash) do bloco anterior e do próprio hash do bloco atual, como na Figura 2.14. Adulterar um bloco alteraria seu próprio sumário criptográfico e exigiria alteração dos blocos subsequentes. Os blocos são propostos por um dos nós participantes da blockchain, o qual pode ser denominado líder ou coordenador em blockchains permissionadas baseadas em protocolos clássicos de consenso, ou mesmo minerador, caso de blockchains não permissionadas baseadas em um desafio computacional.

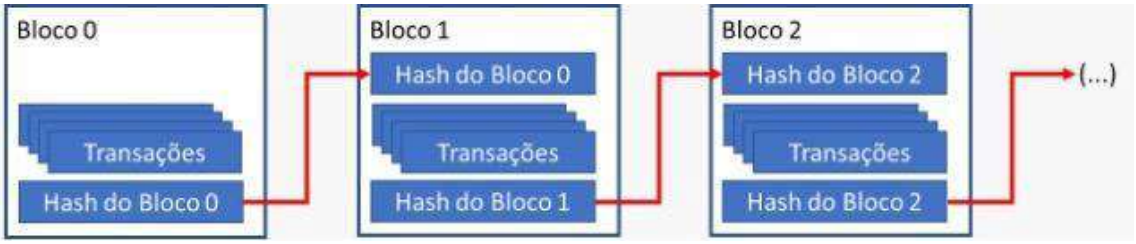

Figura 2.14. Cadeia de blocos em uma blockchain [Freitas et al. 2023].

A proposição de blocos em uma blockchain pública pode envolver mineração, ou seja, há um custo computacional para o proponente do bloco, o qual pode receber como incentivo a criptomoeda associada a blockchain, bem como uma taxa variável por transação.

Nestes cenários, é possível o uso de estruturas ditas *off-chain* que registram uma sequência de chamadas fora da blockchain pública, possibilitando submissão em lote (*batching*) de transações e a redução da taxa paga ao proponente de um novo bloco [Wang et al., 2021]. Uma estrutura *off-chain* pode ser utilizada mesmo para registrar transações até o momento que seja apropriado persistir na cadeia principal, i.e. na blockchain, redes de pagamento, como a *Lighting* [Poon e Dryja 2015], agregam canais de pagamento *off-chain* para registro de transações baseadas em criptomoeda de uma blockchain pública, possibilitando que nós empenhem valores em um canal de pagamento, realizem transações neste canal (e deste interagindo com outros canais de pagamentos pela rede de pagamento) e ao final persistam em uma transação o saldo do conjunto de operações.

Em blockchains públicas, o ponto de decisão para manutenção dos registros *off-chain* ou na blockchain pode utilizar uma política de alto nível, observando, por exemplo, o compromisso entre o nível de latência desejado e o custo associado às transações.

Uma outra abordagem é a percepção de que sendo o consenso o núcleo de operação da Blockchain, o uso de técnicas adaptativas ou autonômicas para o consenso implica na melhoria da operação da blockchain como um todo.

Por exemplo, em [Silva et al., 2023], é demonstrada a relação de compromisso entre diferentes ajustes de tamanho e *timeout* de blocos na plataforma de blockchain Hyperledger Fabric com a capacidade computacional da plataforma e o seu impacto no atraso e vazão do sistema, ou seja, a substituição do protocolo de consenso por uma versão adaptativa (como no caso do PBFT pelo aPBFT explorada na seção anterior), permitiria explorar esta relação de compromisso com ganho ao desempenho da plataforma de blockchain.

Podemos ainda incorporar mecanismos autonômicos à configuração do próprio mecanismo de consenso (e não somente em parâmetros operacionais). Em [Freitas et. al 2023], é proposta a vCubeChain, uma blockchain permissionada que tem por objetivo ser escalável. A vCubeChain é baseada na topologia distribuída hierárquica vCube [Duarte et al. 2014, Ruoso et al. 2014]. A topologia virtual é mantida através de um detector de falhas que forma um hipercubo quando todos os processos estão corretos e que de forma autonômica, na medida em que processos falham por parada, se reorganiza, mantendo diversas propriedades logarítmicas. A vCubeChain é permissionada, i.e. todos os processos são devidamente autenticados. Um líder é eleito utilizando uma estratégia autonômica de difusão confiável para disseminar blocos na rede. Em caso de falsas suspeitas, podem haver múltiplos líderes concorrentes simultaneamente, permitindo a ocorrência de *forks* temporários como em ambientes não permissionados [Nakamoto 2008], mas por meio da cadeia de blocos e de um mecanismo de difusão confiável associado a vCube, a blockchain se mantém íntegra, sempre retornando a um estado consistente de um único líder com a conciliação de divergências.

O entendimento de que a blockchain é uma solução de engenharia que associa ao consenso a estrutura da cadeia de blocos, nos permite pensar em diferentes outras possibilidades a partir da proposição ou da releitura dos algoritmos de consenso, trazendo autogerenciamento a operação destes. Por exemplo, os cenários de blockchain permissionada muitas vezes envolvem máquinas de diferentes organizações, as quais em geral estão dispostas como clusters conectados por meio da Internet. Cada cluster representa um grupo de computadores interconectados por uma rede local, onde pode-se assumir um comportamento síncrono, contudo a nuvem não é em si um sistema síncrono. Este ambiente é representado por um modelo de sistema distribuído híbrido, onde cada cluster local é um sub-sistema síncrono interconectado por canais de comunicação assíncronos com os clusters remotos.

Em um cenário como este, o uso de um consenso adaptativo apresentado em [Gorender, Macêdo e Raynal 2007] permite monitoramento da qualidade de serviço de processos e canais de comunicação, construindo o quórum do consenso adaptado à qualidade de serviço existente no ambiente distribuído e possibilitando o progresso da computação baseado nesta percepção. Ou seja, é possível o progresso mesmo na presença de uma maioria de processos falhos, se as informações do sistema permitem, por exemplo, monitoramento por canais síncronos de todos os processos.

2.4.5 Livro-razão distribuído baseado em reputação

Nesta seção apresentamos como uma blockchain pública pode apresentar maior nível de resiliência por meio de uma abordagem baseada em reputação. Muitos pontos fracos foram associados ao Bitcoin e tecnologias relacionadas, incluindo consistência fraca, baixa taxa de transferência de transações e vulnerabilidade a ataques. Nomeadamente, todas as variantes

contemporâneas baseadas em PoW da Bitcoin dependem da suposição de que um intruso não pode ter mais de 33% ou de 50% do poder de computação a qualquer momento. No entanto, com a sofisticação dos ataques montados no Bitcoin, por exemplo, ataques *flash* (também conhecidos como ataques de suborno), onde um intruso pode obter uma maioria temporária (> 50%) do poder de computação, alugando a capacidade de mineração, todos esses sistemas falham.

Esta seção resume o trabalho apresentado em [Yu et al. 2019], em que são abordadas estas deficiências e apresentadas soluções para as mesmas. Propomos o RepuCoin, o primeiro sistema que pode impedir ataques contra um intruso que pode possuir mais de 50% do poder de computação de toda a rede temporariamente (por exemplo, algumas semanas ou até meses). A prova de conceito mostra que, embora forneça melhores garantias de segurança do que os protocolos anteriores, o RepuCoin também garante uma taxa de transferência muito alta (10.000 transações por segundo - TPS). Na prática, a Visa confirma uma transação em segundos e processa 1.700 TPS em média. Isso mostra que o RepuCoin satisfaz a taxa de transferência necessária de aplicativos do mundo real.

O sistema aborda os desafios acima mencionados definindo um novo princípio de design, chamado prova de reputação. A prova de reputação é baseada na prova de trabalho, mas com duas melhorias fundamentais. Primeiro, sob prova de reputação, o poder de decisão de um minerador (ou seja, o poder de voto para chegar a um consenso no sistema) é dado por sua reputação. A reputação de um minerador não é medida pelo que chamamos de “potência instantânea” do minerador, ou seja, a potência de computação do minerador em um curto intervalo de tempo, como no clássico PoW. Em vez disso, a reputação é calculada com base na quantidade total de trabalho válido que um minerador contribuiu para o sistema e na regularidade desse trabalho, durante todo o tempo durante o qual o sistema esteve ativo. Chamamos isso de "potência integrada" do mineiro. Assim, quando um intruso entra no sistema, mesmo que ele tenha uma capacidade de mineração muito forte, ou seja, alta potência computacional (ou seja, instantânea), ele não teria potência integrada de imediato, ou mesmo logo depois, pois não contribuiu para o sistema antes deste momento de ingresso. Em segundo lugar, quando um minerador se desvia das especificações do sistema, o RepuCoin diminui a reputação do minerador e, portanto, sua potência integrada, em consequência dessa contribuição negativa. Isso evita que um poderoso minerador malicioso ataque o sistema repetidamente sem consequências significativas. Em contraste, os sistemas PoW clássicos não suportam nenhum recurso para punir mineradores que não cumprem as especificações do sistema ou punem esses mineradores simplesmente revogando suas recompensas, ou seja, isto não os impede de atacar o sistema novamente imediatamente depois.

Além disso, o RepuCoin fornece garantias determinísticas em transações, empregando um consenso de voto ponderado baseado em reputação. O consenso é realizado por um grupo formado pelos principais mineradores respeitáveis. Cada membro desse grupo tem um peso associado ao seu voto. O peso do voto de um membro é a porcentagem da reputação desse membro na reputação de todo o grupo. Esses pesos garantem que o poder de voto de alguém dependa não da potência instantânea absoluta (computacional) - que é o facilitador dos ataques de flash - mas da potência integrada, que leva tempo para ser construída e é construída com base na honestidade e no desempenho histórico do minerador.

Na Figura 2.15, apresentamos a análise da segurança fornecida pelos mecanismos usados no RepuCoin em comparativo com outras plataformas de blockchain, mostrando as características determinísticas da taxa de crescimento do poder de decisão, que o fazem resistir a todos os ataques conhecidos à data.

Attacks/Features	BitCoin	BitCoin-NG	ByzCoin	RepuCoin
Double spending attacks	☠	☠	👍	👍
Selfish mining attack	☠	☠	☠	👍
Bribery/flash attack	☠	☠	☠	👍
Eclipse attacks	☠	☠	😐	😐
Non-forkable chain	☠	☠	👍	👍
Liveness	👍	👍	☠	👍
Throughput	7 tps	?	1,000 tps	10,000 tps

Figura 2.15. Análise comparativa de segurança de RepuCoin.

Além disso, mostramos que a rede atinge robustez estocástica muito alta contra os ataques à sua vivacidade ou segurança. Por exemplo, após um único ano de operação, a RepuCoin é resiliente a todos os ataques que comprometem 26%, 33% e 51% dos recursos de computação da rede, mesmo que esse poder permaneça maliciosamente confiscado por quase 100 anos, 2 anos, e 1 ano, respectivamente. Além disso, mesmo que um intruso racional (*for profit*) possa se apoderar de um enorme poder computacional por um período específico (por exemplo, 90% por até 3 meses), ele não quebrará o RepuCoin, devido ao custo de tais ataques tornar a tentativa irracional. Finalmente, fornecemos uma análise de ataques de infiltração irracionais (ex. ciberterrorismo), com uma comparação do custo de atacar diferentes sistemas.

Modelo de Sistema e de Ameaças

RepuCoin é um sistema composto por um número não predeterminado de nós, chamados de mineradores. Cada mineiro tem uma pontuação de reputação, que determina a capacidade desse mineiro de obter recompensas. A pontuação de reputação de um minerador é baseada na correção de seu comportamento e na regularidade em adicionar blocos à cadeia existente, portanto, correlacionada com o poder de computação do minerador. A RepuCoin considera a rede não confiável, e possuindo sincronia parcial. Consideramos um adversário malicioso (também conhecido como Bizantino), que pode atrasar, descartar, reordenar, inserir ou modificar mensagens arbitrariamente. Também consideramos conluios de um número arbitrário de mineradores. No RepuCoin, verificamos e confirmamos microblocos, contidos em *keyblocks* a serem adicionados ao blockchain, usando protocolos de tolerância a faltas Bizantinas com pequenas modificações. Essa forma de acordo evita que um líder mal-intencionado gaste uma moeda em dobro e resolve possíveis bifurcações resultantes de bloqueios de teclas minerados simultaneamente.

Para lidar com as incertezas e ataques acima mencionados o RepuCoin conta com a noção de ter um grupo de consenso, denotado por X, capaz de controlar as operações da RepuCoin, ou seja, executar o protocolo de consenso sobre as transações, orquestrado por um conjunto de mineradores com reputação cumulativa esmagadora. O grupo de consenso pode ser infiltrado por adversários. Por isso, as decisões são tomadas por votação. As regras de votação no consenso não são nominais, mas baseadas em uma votação ponderada inovadora, com base na reputação. Assumimos assim que o adversário consegue controlar no máximo y membros do grupo cuja reputação coletiva é inferior a $1/3$ da reputação cumulativa dos membros do grupo de consenso X. Em consequência, refinamos a definição de quóruns da seguinte forma: chegar a um acordo não requer apenas votos de $2f + 1$ nós, mas também exige

que esses nós tenham coletivamente mais de *2/3* da reputação cumulativa do grupo de consenso. Sob essa suposição, o sistema está seguro e vivo (*safe and live*).

Sistema e função de reputação

Pretendemos definir os objetivos sociais de reputação na RepuCoin de forma precisa e parametrizável, mediante uma função que avalia o andamento da reputação de cada minerador. Esses objetivos são: (i) partida cuidadosa, por meio de um aumento inicial lento; (ii) potencial de recompensa rápida de participantes maduros, por meio de aumento rápido na meia-idade; (iii) prevenção do excesso de controle, por aumento lento próximo ao topo.

No entanto, os mineradores podem ter mau comportamento. Diz-se que um minerador se comporta mal se apresentar mensagens assinadas conflitantes para outros membros do grupo de consenso; ou, se confirmar microblocks com transações conflitantes quando o minerador for eleito líder. Após sua ocorrência, uma evidência de tal mau comportamento é incluída pelos mineradores honestos no blockchain como uma transação especial, e o minerador faltoso perde sua reputação de imediato.

Com base na forte garantia determinística derivada da votação ponderada com base na reputação, a robustez do RepuCoin cresce com o tempo de operação legítimo: quanto mais tarde o intruso entrar, mais seguro será o sistema. Por exemplo, um intruso que ingresse no sistema após um ano de operação, precisaria de pelo menos 51% do poder total de computação e precisaria se comportar corretamente no sistema por 10 meses antes de conseguir fazer com que o RepuCoin perdesse a vitalidade. Quebrar a segurança do RepuCoin é pelo menos tão difícil quanto quebrar sua vivacidade.

2.4.6. Ecossistemas de veículos autônomos (direção de carros autônomos)

Sistemas autônomos cooperativos, como veículos terrestres (carros) em sistemas rodoviários abertos, têm usado ampla tolerância a falhas, por exemplo, em funções x-by-wire, e são bastante seguros do ponto de vista de falhas acidentais. No entanto, eles apresentam desafios a serem tratados devido a falhas maliciosas. Por exemplo, os famosos modelos Tesla FSD foram recentemente sujeitos a vários ataques cibernéticos bem-sucedidos, alguns colocando a segurança em risco.

Essas ameaças incluem todo o ecossistema, desde sistemas e redes veiculares, infraestruturas rodoviárias até redes de comunicação V2V e V2I, a exemplo das vulnerabilidades listadas abaixo:

1. Hackers podem tentar obter acesso não autorizado aos sistemas de controle de veículos autônomos, comprometendo sua segurança e funcionalidade. Nesses casos, são necessárias medidas robustas de cibersegurança, como criptografia, protocolos de comunicação seguros, sistemas de detecção de intrusão e auditorias de segurança regulares para detectar e prevenir ataques cibernéticos;
2. Veículos autônomos dependem fortemente de sensores e sistemas de percepção para coletar informações sobre o ambiente ao redor. Esses sistemas podem ser suscetíveis a erros ou mau funcionamento, levando a interpretação incorreta de dados e situações potencialmente perigosas. Para esses casos, podem ser utilizados sistemas de redundância e backups para sensores críticos;
3. Veículos autônomos coletam grandes quantidades de dados, incluindo geolocalização, padrões de direção e informações pessoais de passageiros. Se esses dados caírem em

mãos erradas, podem ser usados para roubo de identidade, vigilância ou ataques direcionados. Para esses casos, são aplicadas técnicas de criptografia de dados, com consentimento explícito para coleta e uso de dados, para proteger a privacidade do usuário.

Por essas razões, a resiliência cibernética se apresenta como uma facilitadora da operação segura para os sistemas cooperativos autônomos, em particular, os automotivos.

2.4.7. Ecossistemas de veículos autônomos

Quase todos os fabricantes de automóveis experimentam veículos totalmente autônomos e começam a acumular quilômetros em estradas e rodovias para seus *safety cases*, visando aumentar progressivamente o nível de autonomia. Mas não menos importante, a conectividade também vem aumentando, por motivos como infoentretenimento, assistência de trânsito, manutenção remota ou localização de veículos. A este cenário acresce oportunidades de cooperação entre estes veículos, não apenas para melhorar a funcionalidade autônoma, mas também, e principalmente, para a segurança na condução.

Em [Lima et al. 2016], explica-se a problemática dos veículos terrestres autônomos, com ênfase num conjunto de pontos-chave: (a) a necessidade de considerar a autonomia de veículos automóveis como um todo, isto é, um ecossistema completo orientado para veículos autônomos e cooperativos, incluindo os veículos autônomos, as redes e sistemas de periferia (*road-side units, edge web and cloud*), o ambiente não habilitado, como veículos clássicos e pessoas, etc.; (b) a compreensão do aumento dramático da superfície de ameaça global que surge desta nova realidade, com ênfase para a segurança informática; (c) a introdução de um hiato cultural para a indústria, o *safety-security gap*, ameaçando os níveis de *safety* por via de uma compreensão imperfeita dos riscos de *security*.

O principal objetivo desse documento de posição é assim duplo: (i) propor um modelo holístico e genérico bastante, de ecossistema de veículos autônomos e cooperativos; e (ii) realizar uma análise do plano de ameaças, detalhando vetores de ameaças específicos aos quais os designers devem estar atentos, bem como propondo mitigações para os mesmos.

Acresce ainda que a conectividade e cooperação se tornarão a norma, e prevemos que, além dos carros, ela será estendida a, por exemplo, pedestres, bicicletas, etc., formando um universo de objetos 'sencientes', no sentido informático do termo. Nesse sentido, o ecossistema de veículos autônomos e cooperativos pode em breve emergir como uma infraestrutura de informação crítica (virtual) (CII) com grande importância social. Como consequência, a natureza e a intensidade das próprias ameaças também deve evoluir. Além de falhas acidentais e ataques de hackers casuais, devemos esperar ameaças persistentes avançadas e ataques direcionados montados por adversários altamente qualificados e bem equipados, o que reforça a atualidade deste estudo. Começamos por recordar os vários acidentes (alguns mortais) com carros autônomos da Tesla, alegadamente devido a falhas de segurança no sistema de controle da condução autônoma. Mas para além de ataques a carros bem-sucedidos e já demonstrados, comprometendo a comunicação celular e Wi-Fi para desligar o motor, desabilitar freios e trancar as portas, imagine tentativas coordenadas de intrusão em carros autônomos para causar acidentes graves, ou usá-los para bloquear estradas para ações criminosas ou terroristas. Não são infelizmente cenários de ficção os casos relatados no estudo.

A superfície de ameaças do ecossistema de veículos autônomos

No que respeita à conectividade, existem vários níveis de abertura nas redes envolvidas, que vão desde: a Internet, que é pública; as intranets RSU(*road-side unit*), que são privadas, mas expostas geograficamente, exibindo conexões sem fio; as redes V2V e V2I (*vehicle-to-vehicle, vehicle-to-infrastructure*), que são públicas e sem fio; redes veiculares sem fio; e as menos acessíveis, redes a cabo para carros, exigindo acesso físico ou um primeiro salto através da barreira entre o *gateway* de rede sem fio do carro e a infraestrutura de rede veicular restante - ver Figura 2.16. Assumimos que as ameaças podem ser efetuadas por quatro categorias de *agentes de ameaças*: Externo (X): por exemplo, computadores na Internet, RSUs comprometidos, s-Cars (veículos sencientes) hostis ou computadores hostis próximos a s-Cars; Local (L): computadores comprometidos ou hostis dentro de s-Cars, mídia conectada (por exemplo, pen-drives não autorizados). Físico (P): computadores comprometidos ou hostis via tomadas de manutenção, humanos desonestos substituindo ou inserindo hardware; subversão da cadeia de fornecimento. Ambiental (E): dispositivos que interferem nas propriedades do ambiente físico (por exemplo, bloqueadores que perturbam as transmissões sem fio ou dispositivos semelhantes que levam à criação de falsas imagens do sensor); RSUs falsos.

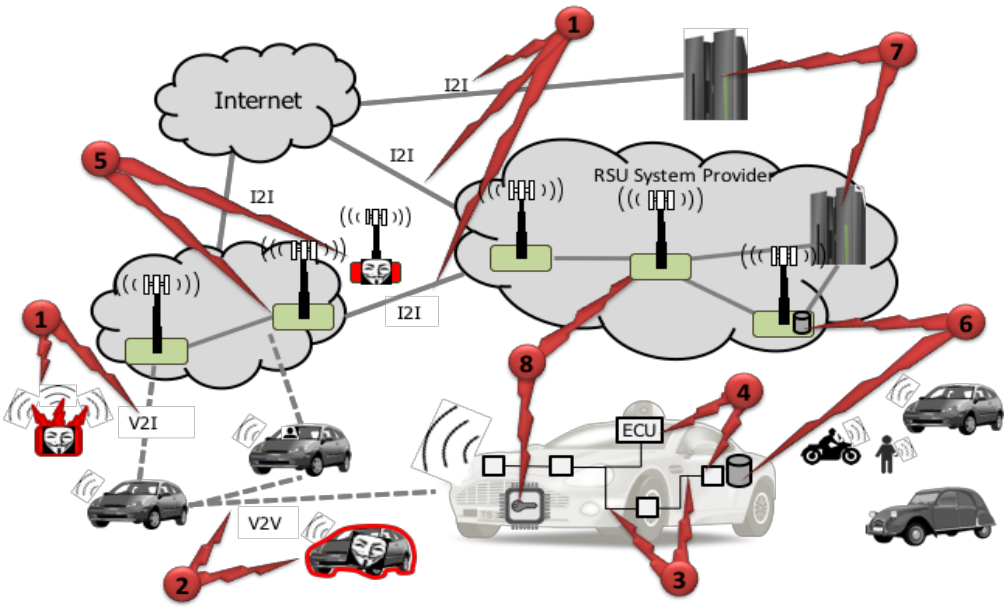

Figura 2.16. Ecossistema de Veículos Autônomos.

Essas ameaças são materializadas por meio de diversos vetores, exemplificados acima e listados abaixo - ver Figura 2.17, com menção aos possíveis agentes em cada vector.

Vector	Description	Agents
#1	Attacks on global V2I/I2I communication infrastructure	X, L, E
#2	Attacks on local V2V communication infrastructure	X, L, E
#3	Attacks on in-vehicle communication infrastructure	L, P
#4	Attacks on vehicle computing nodes' software	L,P
#5	Attacks on road-side units'software	X, P, E
#6	Attacks on sensors and control-sensitive data	X, L, P, E
#7	Attacks on authentication mechanisms	X, L, P
#8	Physical-level attacks	P

Figura 2.17. Vetores de ameaças e agentes correlacionados.

Rumo a Ecossistemas de Veículos Autônomos e Cooperativos Seguros e Protegidos

Nesta secção, resumem-se os trabalho apresentados em [Vöelp e Verissimo 2018; Shoker e Verissimo 2022], em que se tenta dar uma resposta à questão de como a *safety* e a *security* podem ser simultaneamente garantidas por métodos automáticos, fechando o hiato *safety-security*.

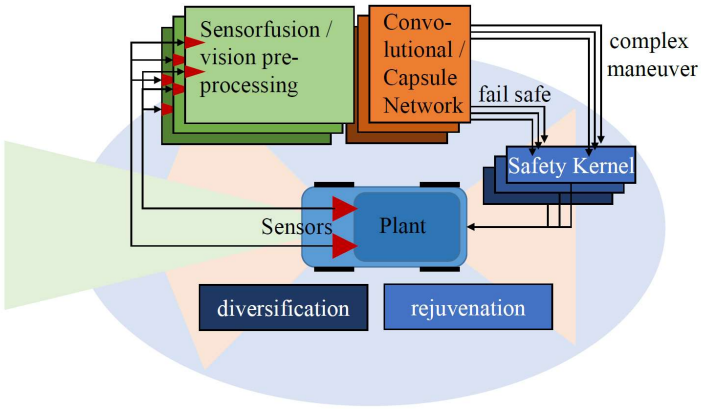

Figura 2.18. Diagrama de fluxo e componentes essenciais da arquitetura [Vöelp e Veríssimo 2018].

A Figura 2.18, do trabalho em [Vöelp e Verissimo 2018], esboça o diagrama de fluxo e os componentes essenciais de nossa arquitetura tolerante a intrusões para condução autônoma. A plataforma é construída em torno do conceito de hibridização arquitetônica. Instanciações prévias do conceito no contexto CPS são, por exemplo, a arquitetura *Simplex* ou a arquitetura *Timely Computing Base*. A arquitetura segue portanto um modelo de falha híbrido, que neste trabalho atual é estendido da *safety* para a *security*. Baseamo-nos na extensão para CPS de trabalhos anteriores em tolerância a intrusões, como a arquitetura *Trusted Timely Computing Base*. Na figura, um controlador simples mas altamente confiável— o kernel de segurança — decide em última análise, caso necessário, acerca da execução correcta das manobras propostas normalmente pelos subsistemas complexos de visão e de IA, de maneira atempada e segura, dependendo das condições de funcionamento em cada momento. A simplicidade exigida pelo kernel de segurança é um artefacto da solução: deriva, de facto, da necessidade óbvia de verificabilidade, de modo a atingir um design ultra correcto e assim justificar a hipótese de falha híbrida — o kernel deve ter uma probabilidade de falha infinitamente menor que a do sistema principal.

Em [Shoker e Verissimo 2022], apresentamos uma evolução e generalização destas aproximações, através do conceito de *Intrusion Resilience Systems* (IRS) para veículos autônomos modernos. O IRS visa contribuir para uma revolução disruptiva nas arquiteturas atuais de computadores e redes de veículos, estendendo as propriedades de segurança e proteção de arquiteturas baseadas em componentes (por exemplo, AUTOSAR). Propomos aproveitar a corrente maturidade do mercado de ECUs, que oferece hoje em dia uma diversidade e independência de oferta, a qual irá permitir propor arquiteturas tolerantes a faltas e intrusões implementadas por SW, de forma semelhante aos princípios estabelecidos pelas arquiteturas pioneiras de TI dos anos 80 em tolerância a faltas distribuída, modular e incremental. Os novos conceitos avançados pelo middleware automotivo IRS, permitem a execução de múltiplas e possivelmente diversas réplicas de um processo de aplicação com estado completo em diferentes ECUs, formando uma Máquina de Estado Replicada

determinística resiliente. Aplicativos distribuídos como fechaduras de portas, controle de janelas, atualizações de software Over-the-Air (OTA) são alguns exemplos de aplicativos viáveis no topo do IRS.

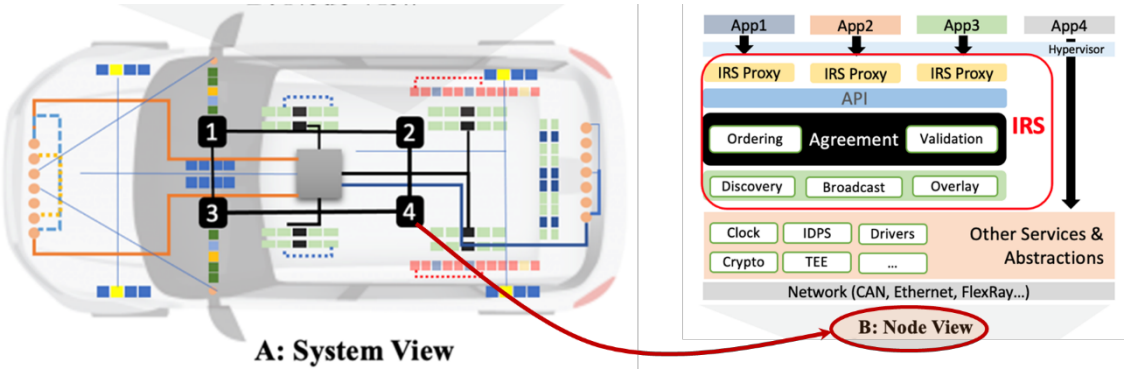

Figura 2.19. Arquitetura do sistema IRS.

Modelo e Arquitetura do Sistema

Apresentamos uma vista da arquitetura do sistema IRS na Figura 2.19. No lado esquerdo a vista global mostra um sistema veicular com um número N de nós IRS ($N = 4$, neste caso) replicados sobre N ECUs. Um nó é composto por um dispositivo de computação, ou seja, uma ECU, uma pilha de software correspondente e um aplicativo veicular de tempo real (crítico) para simplificar. Um nó pode se comunicar com suas contrapartes por meio de mensagens por meio de uma rede veicular, por meio de um link direto, um comutador ou um gateway. Para maior clareza, usamos Unidades de Controle Zonal (ZCU) como ECUs para hospedar diferentes aplicações (por exemplo, fechaduras de portas e controle de janelas) na mesma ECU. Por outro lado, o lado direito da figura apresenta a vista interna de um dos nós (ou seja, o nó 4) descrevendo seus componentes e relação dentro da pilha de *hardware/software* (HW/SW).

Um aplicativo pode falhar ou se comportar de forma arbitrária ou maliciosa quando sujeito a uma intrusão. Assumimos que no máximo uma fração F de N nós pode falhar por vez, e que existe independência de falhas entre os nós. Isso pode ser alcançado empregando ECUs de diversos fornecedores, diferentes bibliotecas, pilha de software e implementação, etc., o que não é incomum no cenário automotivo. O conceito do IRS é baseado na ideia de mascaramento de erro de intrusão em vez de detecção e prevenção como no IPS/IDS. Ao executar várias (N) réplicas/versões de um aplicativo e comparar suas saídas em diferentes nós (ECUs), é possível mascarar qualquer erro causado por falhas acidentais ou maliciosas que ocorrem em nós F com falha, adotando o estado de saída de um não infectado maioria ($N - F$). Nesta abordagem, o estado de uma aplicação crítica só pode ser modificado com a concordância de pelo menos N-F contrapartes.

2.4.8. Redes veiculares autônomas sensíveis ao contexto

As Redes Veiculares Ad-Hoc (VANETs) apresentam como grande motivador a possibilidade de viabilizar o desenvolvimento e execução de aplicações veiculares orientadas para a segurança no trânsito e para o tráfego inteligente. Estas aplicações poderão, por exemplo, gerenciar cruzamentos e semáforos evitando acidentes, assim como controlar o tráfego para

que este tenha um fluxo mais eficiente. Estas possibilidades fazem das VANETs um relevante tópico de pesquisa, com novos resultados sendo continuamente apresentados.

Estas redes apresentam uma topologia dinâmica, sendo construída e modificada a todo momento, devido à alta mobilidade dos veículos. Manter a comunicação nestes casos, e descobrir rotas de comunicação que possam se manter estáveis por mais tempo, é um desafio. Protocolos para VANETs precisam de informações atualizadas sobre o ambiente de comunicação e sobre as próprias aplicações, para ser capaz de prover um serviço de comunicação com o máximo de eficiência possível. Uma maneira de se formalizar estas informações é através do conceito de sistemas cientes de contexto.

Em sistemas cientes do contexto (Context-Aware Systems -- CAS) informações coletadas do ambiente são utilizadas pelo sistema para tomar decisões relativas à sua execução, adaptando seu comportamento a alterações em diferentes aspectos deste ambiente. Desta forma, o sistema estará sempre monitorando e coletando novas informações de contexto para serem utilizadas nas suas decisões [Schilit e Theimer 1994]. O conceito de sistemas cientes do contexto foi inicialmente proposto para sistemas pervasivos e sistemas baseados em sensores, nos anos 90 descrevendo diferentes aspectos e tipos de contexto e seu uso por sistemas.

Pode-se considerar que Contexto é qualquer informação que pode ser usada para caracterizar a situação de entidades ditas relevantes para os sistemas e seus ambientes de execução, incluindo informações dos próprios usuários e das aplicações, assim como informações sobre o estado do meio físico nos quais estes sistemas executam.

O conceito de contexto tem sido utilizado em redes veiculares ad hoc (VANETs) para fornecer diferentes tipos de informações aos sistemas.. Nestas redes, o contexto pode ser relativo a aspectos da mobilidade dos veículos, da comunicação, das mensagens sendo trocadas, das aplicações executadas ou dos próprios usuários e motoristas [Vahdat-Nejad et al. 2016].

Em [Sá e Gorender 2019] apresentamos um contexto de comunicação para VANETs. Este contexto é construído a partir da monitoração de informações de mobilidade e comunicação da rede, tais como a última posição conhecida dos veículos, sua velocidade e direção de sua movimentação. Estas informações são coletadas localmente por cada veículo, através de sensores tais como GPS, velocímetro e acelerômetro, ou são calculadas a partir das informações coletadas. Os veículos trocam suas informações de contexto, utilizando um sistema de monitoração de contexto. Cada veículo mantém o seu contexto sobre o ambiente de comunicação, relacionando suas informações locais com informações de outros veículos.

A partir das informações sendo mantidas, são definidos o contexto de vizinhança e o contexto de comunicação.

O contexto de vizinhança pode assumir um entre dois valores:

- Vizinho: um veículo é considerado ser vizinho do veículo local quando ambos estão dentro dos seus alcances de comunicação sem fio; e
- Não-Vizinho: os veículos estão fora dos seus alcances de comunicação.

Este contexto é definido a partir da distância entre os veículos, calculada a partir da última posição disponível dos veículos, e do alcance das antenas de comunicação.

O contexto de comunicação pode assumir os seguintes valores:

- Comunicável: um veículo remoto é considerável comunicável com relação ao veículo local se são vizinhos, ou se existe uma rota de comunicação entre ambos; e
- Não-comunicável: os veículos não se comunicam.

Assumimos que veículos vizinhos são comunicáveis. Veículos que não são vizinhos podem ser comunicáveis, se for possível construir uma rota de comunicação entre ambos, por meio de nós intermediários, que entre si, em sequência, sejam vizinhos. Para a construção e manutenção destas rotas de comunicação é utilizado um protocolo de roteamento ciente do contexto. Este protocolo atualiza as rotas de comunicação, sob demanda, ao tempo em que o contexto vai sendo atualizado, provendo uma comunicação mais estável e confiável.

O conceito de Qualidade de Contexto (QoC) foi proposto como uma forma de fornecer métricas de qualidade sobre as informações do contexto, para serem também utilizadas pelos sistemas ao tomarem decisões baseadas no contexto [Buchholz et al. 2003]. Este conceito foi também proposto inicialmente para sistemas baseados em sensores, propondo métricas associadas às aplicações e métricas associadas a características dos sensores. Estas métricas podem indicar o quanto uma informação de contexto é acurada, recente e confiável.

Em [Sá e Gorender 2019] e [Sá e Gorender 2021] foram apresentadas métricas de QoC aplicadas ao contexto de comunicação, que podem ser utilizadas pelas aplicações e pelo sistema de comunicação em suas decisões.

A métrica Validity Time (Tempo de Validade) fornece uma janela estimada de comunicação para dois veículos comunicáveis.

A métrica Confidence (Confiança) define um grau estimado de confiança na informação do contexto, considerando o momento em que esta informação foi atualizada e sua idade (Age).

Estas e outras métricas são úteis em definir o quanto o contexto pode ser considerado útil e válido para ser utilizado. Este é um trabalho em andamento no Laboratório de Sistemas Distribuídos (LaSiD) e no Programa de Pós-graduação em Mecatrônica (PPGM), ambos da Universidade Federal da Bahia (UFBA).

2.4.9. Sistemas de Controle de Plantas Elétricas baseados em IoT e Sistemas Ciberfísicos

Nas últimas décadas, as infraestruturas de serviços elétricos tornaram-se amplamente informatizadas, controladas remotamente/automaticamente e interconectadas. Ainda existe a crença de que os sistemas SCADA [Supervisão, Controle e Aquisição de Dados] que controlam essas infraestruturas são legados, fechados e inatacáveis, ou que basta apenas usar um firewall e um detector de intrusão. Esta percepção é inadequada, pelo que é necessário investir em configurações onde a abordagem de prevenção seja complementada com dispositivos de *middleware* que alcancem a segurança automática, através da tolerância a falhas e intrusões. As redes elétricas, se arquitetadas e gerenciadas tendo em mente os mesmos objetivos de segurança e confiabilidade dos sistemas clássicos de tecnologias da informação, podem apresentar níveis muito altos de resiliência.

A resiliência à severidade esperada das ameaças aos sistemas de energia requer mecanismos adicionais que busquem uma operação sustentável e autônoma. Mecanismos de recuperação proativos e reativos para autocorreção são discutidos, bem como mecanismos de monitoramento de confiabilidade que permitem adaptação confiável às situações não previstas ou além das suposições.

O problema de segurança e confiabilidade, ou genericamente falando, resiliência de infraestruturas de *utilities*, isto é, infraestruturas baseadas em sistemas ciberfísicos, não é totalmente compreendido, principalmente devido à composição híbrida destas infraestruturas. O controle do processo das infra-estruturas das concessionárias assenta nos sistemas SCADA (*Supervisory Control and Data Acquisition*) que conferem a capacidade operacional de aquisição de dados, supervisão e controle seja qual for o negócio em causa (eletricidade, água, gás, telecomunicações). No entanto, eles também têm interconexões com as intranets corporativas padrão e, portanto, indiretamente com a Internet. Os sistemas SCADA mencionados acima não foram projetados para serem amplamente distribuídos e acessados remotamente, muito menos para serem abertos. Eles cresceram fechados, sem pensar na *security*. Focando na rede elétrica, as redes de produção, transmissão e distribuição estão hoje em dia largamente automatizadas nos aspectos funcionais, e abertas à rede e à utilização de HW/SW *standard* de mercado (*COTS*), mas carecem de uma aproximação robusta à resiliência, isto é, a garantia combinada de *safety* e *security*.

As perspectivas de danos que podem resultar desta exposição são esmagadoras. Eles vão desde manobras erradas, até ações maliciosas vindas de terminais localizados fora, em algum lugar da Internet. Os alvos dessas ações são unidades de controle de computador, componentes e sistemas embarcados, ou seja, dispositivos conectados ao hardware operacional (por exemplo, bombas e filtros de água, geradores de energia elétrica e proteções de energia, comportas de barragens, etc.). Esta situação foi, por exemplo, analisada sistemicamente no projeto CRUTIAL [Dondossola et al. 2006], onde foram dados vários exemplos de falhas (acidentais ou maliciosas) em sistemas de energia. Essas falhas têm-se repetido ao longo dos anos até recentemente.

As infraestruturas críticas enfrentam um risco visivelmente alto. Possíveis ameaças incluem extorsão, terrorismo e ataques patrocinados pelo governo. Infelizmente, os níveis de vulnerabilidade dessas infraestruturas são altos. Estas infraestruturas críticas combinam computação com processos físicos ou mecânicos controlados eletronicamente por sistemas (SCADA, PCS) que usualmente têm fraca *security*, sendo muitos sistemas legados proprietários com alguma antiguidade. Correntemente, estes encontram-se combinados com sistemas recentes standard de mercado. Em suma, a desculpa comum, “os hackers não conhecem nossos sistemas”, não é mais verdadeira.

Resumimos nesta seção o trabalho apresentado em [Bessani et al., 2008], onde se descrevem os resultados do projeto Europeu CRUTIAL, Critical Utility Infrastructural Resilience, com particular destaque para uma nova arquitetura e protocolos que preservam os sistemas legados e para um novo dispositivo que fornece proteção incremental, garantindo diferentes níveis de resiliência aos diferentes partes da infraestrutura, de acordo com sua criticidade.

A aborgagem CRUTIAL à proteção de infraestruturas críticas ciberfisicas

Um dos pontos cruciais do problema de proteção de infraestrutura crítica é a segurança das interconexões entre provedores de infraestrutura, reguladores, operadores e outros. Para enfrentar esse problema, precisamos de uma arquitetura que nos permita modelar e raciocinar sobre essa realidade. No Crutial, toda a arquitetura da infraestrutura é representada como uma rede WAN-of-LANs, isto é, uma rede pública (WAN) de redes locais (LANs). Este modelo representa bem uma típica infraestrutura de informação crítica, como ilustrado na Figura 2.20.

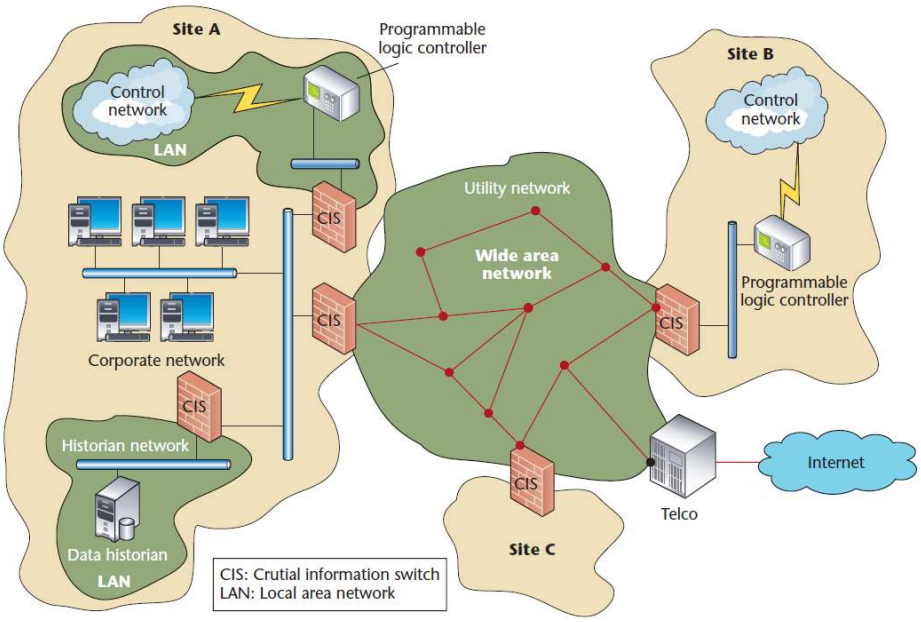

Figura 2.20. Infraestrutura típica de informação crítica.

Essa arquitetura nos permite definir domínios com diferentes níveis de confiabilidade, permitindo defender os domínios uns dos outros — ou seja, uma LAN de outra LAN ou da WAN. Um comutador de informações (CIS) protege cada LAN (ver acima) e fornece dois serviços básicos: (1) o serviço de proteção garante que o tráfego de entrada e saída da LAN satisfaça a política de segurança da organização que gerencia a LAN; (2) o serviço de comunicação que suporta comunicação segura entre os CIS das várias LANs e, em última análise, entre os dispositivos locais e o exterior.

Um CIS é um dispositivo de proteção distribuído com as seguintes características principais: (i) assemelha-se a um *firewall* distribuído porque as organizações podem implantá-lo não apenas na borda da rede, mas também dentro da rede para proteger equipamentos críticos; (ii) interpreta um modelo elaborado de controle de acesso; (iii) é tolerante a intrusões, ou seja, funciona corretamente mesmo que um intruso invada algum de seus componentes, sendo seu objetivo resistir a um alto grau de hostilidade do ambiente. Comparado a outros trabalhos tolerantes a intrusões, o CIS assegura a integridade de funcionamento dos dispositivos legados eventualmente a seu jusante (dentro das LANS) contra aos ataques que possam ser perpetrados por réplicas comprometidas do CIS que não satisfaçam a política de segurança (ver mais adiante). Do ponto de vista funcional, o serviço de proteção CIS funciona como um firewall, interposto entre a WAN e os dispositivos vulneráveis de uma LAN, como ilustra a Figura 2.22.

Designs incrementais da resiliência do CIS

Dados os vários níveis de criticidade de equipamentos de infraestrutura crítica e o custo de usar um dispositivo replicado, vale a pena definir uma hierarquia de projetos de CIS incrementalmente mais resilientes.

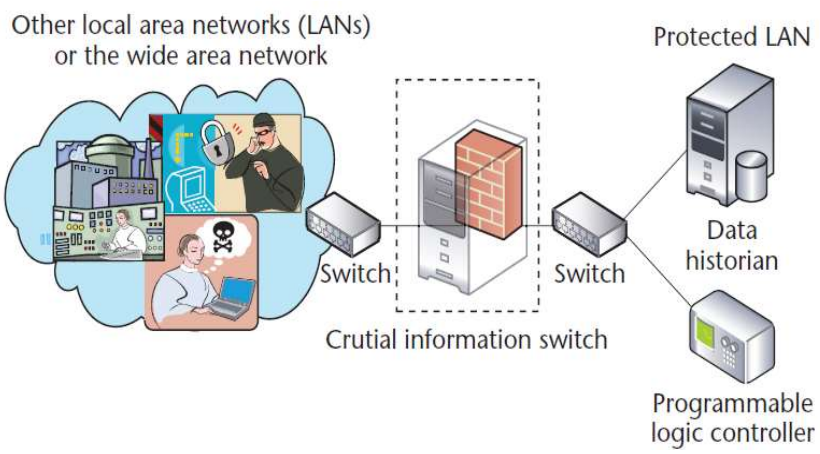

Figura 2.21. Serviço de proteção CIS.

CIS não tolerante a intrusões: a Figura 2.21 sugere um CIS não tolerante a intrusões. É o projeto mais barato porque requer apenas uma máquina, assim como um firewall clássico. Oferece melhor proteção do que os firewalls normais contra ataques de *bypass*, usando a aplicação de políticas no nível do aplicativo e um modelo de controle de acesso avançado. No entanto, é vulnerável a ataques diretos ao próprio CIS.

CIS tolerante a intrusões: o projeto de um CIS tolerante a intrusões consiste em utilizar *2f + 1* máquinas processando os pedidos de tráfego em paralelo, para tolerar intrusões em f réplicas do CIS - ver Figura 2.22. Um pedido chega a todas pela utilização, por exemplo, de um *hub Ethernet*. Os resultados de todas as réplicas são votados, e passa o resultado majoritário, isto é, se e somente se pelo menos f + 1 réplicas votarem a favor: deixar passar a mensagem original (e não uma modificada), ou o bloqueio da mesma. O CIS então transmite uma mensagem aprovada para o destino usando uma réplica distinta, o líder, para que nenhuma multiplicação de tráfego desnecessária ocorra dentro da LAN.

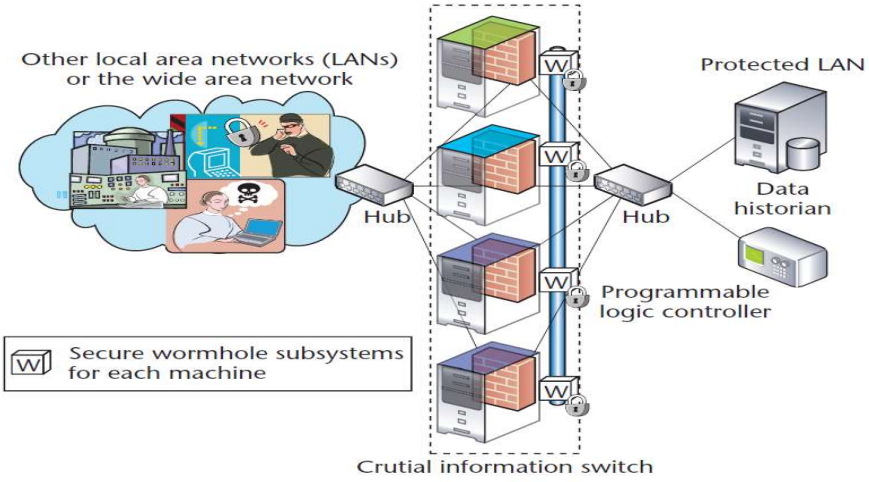

Figura 2.22. CIS tolerante a intrusões.

CIS tolerante a intrusões e autorreparável (self-healing): O mais resiliente projeto de CIS, ilustrado na figura acima, combina tolerância a intrusões com mecanismos de autorreparação para lidar com algumas limitações já descritas. Mecanismos de autorreparação usam um

serviço de recuperação proativo-reativo que combina rejuvenescimentos periódicos acionados por tempo com rejuvenescimentos acionados por eventos quando detectam ou suspeitam de um ataque. Este último mecanismo resolve o problema de ataques de réplicas contaminadas, a dispositivos das LANs. Para isso, a arquitetura híbrida apresentada acima possui uma rede simples de comunicação de controle entre os híbridos de todas as réplicas (nós W, *security kernels*), por onde eles podem trocar alarmes e chegar a consensos simples. Por exemplo, se um intruso comprometer uma réplica do CIS e fizer uma ação detectável (por exemplo, enviar uma mensagem de ataque a um dispositivo frágil da LAN), as outras réplicas (correctas) vão detectar essa acção, trocam mensagens, chegam a consenso sobre a réplica comprometida, e a recuperação reativa da mesma é imediatamente disparada pelo seu híbrido W, que rejuvenesce a réplica. Por outro lado, recuperações proativas são acionadas periodicamente em todas as réplicas, mesmo que não estejam comprometidas. O objetivo é remover os efeitos de ataques maliciosos ou falhas, mesmo que o intruso permaneça inativo (furtivo). Este projeto CIS, como se vê na figura, em vez das clássicas $2f + 1$ réplicas, requer $2f + k + 1$ máquinas para tolerar f intrusões por período de recuperação (ditado pela periodicidade das recuperações proativas). O novo parâmetro k representa o número de réplicas que se recuperam ao mesmo tempo. Seu valor é tipicamente um. Se não existisse k, o CIS poderia ficar indisponível durante as recuperações, ou vulnerável a ataques por exaustão de recursos de redundância.

2.5. Conclusões

Os sistemas distribuídos físico-digitais, também conhecidos como sistemas ciberfísicos, surgiram como resultado da distribuição geográfica e interconexão de componentes físicos por meio de redes de comunicação. Esses sistemas abrangem uma ampla variedade de dispositivos que interagem com o ambiente físico, incluindo sensores, robôs, drones e câmeras, e encontram aplicações em setores diversos, como manufatura, transporte, saúde e segurança. No entanto, a relação físico-digital nessas aplicações apresenta desafios significativos, sobretudo em relação à confiabilidade e segurança dos sistemas envolvidos.

Para enfrentar esses desafios, a gestão autônoma desses sistemas oferece uma solução promissora. Ao configurar os sistemas para monitorar seus próprios estados e tomar decisões com base nos dados coletados, eles podem se adaptar e responder efetivamente a condições em constante mudança. O objetivo deste curso é fornecer uma abordagem teórico-prática abrangente para a construção de sistemas distribuídos autônomos ou autonômicos capazes de atender aos requisitos de desempenho, confiabilidade e segurança dos sistemas físico-digitais distribuídos. Compreender a crescente complexidade desses sistemas e adotar estratégias de gestão adequadas é crucial para garantir sua eficiência e eficácia.

Em conclusão, este curso visa capacitar os estudantes com o conhecimento e as habilidades necessárias para enfrentar as complexidades dos sistemas distribuídos físico-digitais, em especial nos aspectos de confiabilidade e segurança. Por meio da exploração de conceitos teóricos e técnicas de implementação práticas, os estudantes terão a oportunidade de adquirir *insights* valiosos nesses campos. Espera-se que, ao final do curso, os estudantes possam contribuir para a implementação bem-sucedida de sistemas físico-digitais distribuídos em diversos domínios.

Referências

[Androulaki et al. 2018] Androulaki, E. et al. Hyperledger fabric: A distributed operating system for permissioned blockchains. In Proc. 13th EuroSys Conference, EuroSys ’18. ACM. 2018.

[Avizienis et al. 2004] Avizienis, Algirdas et al. Basic concepts and taxonomy of dependable and secure computing. IEEE transactions on dependable and secure computing, v. 1, n. 1, p. 11-33, 2004.

[Baker et al. 2002] Baker, Mark, Rajkumar Buyya, and Domenico Laforenza. Grids and Grid technologies for wide‐area distributed computing. Software: Practice and Experience, v. 32, n. 15, p. 1437-1466, 2002.

[Bertier et al. 2002] Bertier, Marin; Marin, Olivier; Sens, Pierre. Implementation and performance evaluation of an adaptable failure detector. In: Proceedings International Conference on Dependable Systems and Networks. IEEE, 2002. p. 354-363.

[Bessani et al., 2008] Bessani, A; Sousa, P; Correia, M.; Neves, N.; Verissimo, P. The CRUTIAL Way of Critical Infrastructure Protection. IEEE Security and Privacy, vol. 6, no. 6, pp. 44-51, Nov/Dec 2008., Dec. 2008.

[Birman 1993] Birman, K. P. The process group approach to reliable distributed computing. Communications ofthe ACM, ACM, New York, NY, USA, v. 36, n. 12, p. 37–53, December 1993.

[Buchholz et al. 2003] Buchholz, T., Küpper, A. and Schiffers, M. “Quality of context: What it is and why we need it,” in Proceedings of the workshop of the HP OpenView University Association, vol. 2003, 2003.

[Castro e Liskov 2002] Castro, Miguel; Liskov, Barbara. Practical Byzantine fault tolerance and proactive recovery. ACM Transactions on Computer Systems (TOCS), v. 20, n. 4, p. 398-461, 2002.

[Chandra e Toueg 1996] Chandra, Tushar Deepak; Toueg, Sam. Unreliable failure detectors for reliable distributed systems. Journal of the ACM (JACM), v. 43, n. 2, p. 225-267, 1996.

[Chandra et al. 1996] Chandra, T. D. et al. On the impossibility of group membership. In: Proc. of the 15th annual ACM Symposium on Principles of Distributed Computing. New York, NY, USA: ACM, 1996.

[Chen et al. 2002] Chen, W., Toueg, S., and Aguilera, M. K. On the quality of service of failure detectores. IEEE Trans. On Computer, 51(2):561–580. 2002.

[Chockler et al. 1998] Chockler, Gregory V., Nabil Huleihel, and Danny Dolev. An adaptive totally ordered multicast protocol that tolerates partitions. In: Proceedings of the seventeenth annual ACM symposium on Principles of distributed computing. 1998. p. 237-246.

[Chockler et at. 2001] Chockler, G. V.; Keidar, I.; Vitenberg, R. Group communication specifications: a comprehensive study. ACM Computing Surveys, ACM, New York, NY, USA, v. 33, n. 4, p. 427–469, December 2001.

[Cristian 1991] Cristian, Flavin. Understanding fault-tolerant distributed systems. Communications of the ACM, v. 34, n. 2, p. 56-78, 1991.

[Cristian 1996] Cristian, F. Synchronous and asynchronous group communication. Communications of the ACM, ACM, New York, NY, USA, v. 39, n. 4, p. 88–97, April 1996.

[Cristian e Fetzer 1999] Cristian, Flaviu; Fetzer, Christof. The timed asynchronous distributed system model. IEEE Transactions on Parallel and Distributed systems, v. 10, n. 6, p. 642-657, 1999.

[Dai e Wang 1992] Dai, Shu-Ho; Wang, Ming-O. Reliability analysis in engineering applications. Van Nostrand Reinhold Company, 1992.

[de Sá 2011] de Sá, Alirio Santos. Mecanismos Autonômicos de Tolerância a Falhas para Sistemas Distribuídos. Tese de Doutorado, Universidade Federal da Bahia. 2011.

[de Sá e Macêdo 2006] de Sá, Alirio Santos; Macêdo, R. J. de A. . Avaliando o Impacto de Detectores de Defeitos na Estabilidade de Sistemas de Controle de Tempo Real sobre Redes Convencionais. In: WTF 2006 - VII Workshop de Testes e Tolerância a Falhas, 2006, Curitiba. Anais do WTF 2006. Curitiba: SBC, 2006. v. 1. p. 55-66.

[de Sá e Macêdo 2010] de SÁ, Alirio Santos; Macêdo, R. J. de A.. QoS Self-configuring Failure Detectors for Distributed Systems. In: DAIS. 2010. p. 126-140.

[de Sá et al. 2013] de Sá, Alirio Santos; Freitas, Allan Edgard Silva; Macêdo, Raimundo José de Araújo. Adaptive request batching for byzantine replication. ACM SIGOPS Operating Systems Review, v. 47, n. 1, p. 35-42, 2013.

[Défago et al. 2004] Défago, X.; Schiper, A.; Urbán, P. Total order broadcast and multicast algorithms: Taxo- nomyand survey. ACM Computing Surveys, ACM, New York, NY, USA, v. 36, n. 4, p. 372–421, December 2004.

[Dolev et al. 1987] Dolev, Danny; Dwork, Cynthia; Stockmeyer, Larry. On the minimal synchronism needed for distributed consensus. Journal of the ACM (JACM), v. 34, n. 1, p. 77-97, 1987.

[Dondossola et al. 2006] Dondossola, G; Deconinck, G; Giandomenico, F; Donatelli, S; Kaaniche, M; Verissimo, P. Critical Utility Infrastructure Resilience. Workshop on Security and Networking in Critical Real-Time and Embedded Systems (CRTES'06), @ RTAS'06, April 2006, San Jose, California, USA.

[Duarte et al. 2014] Duarte Jr, E., Bona, L., and Ruoso, V. VCube: A provably scalable distributed diagnosis algorithm. In 5th ScalA Workshop, 2014.

[Dwork et al. 1988] Dwork, Cynthia; Lynch, Nancy; Stockmeyer, Larry. Consensus in the presence of partial synchrony. Journal of the ACM (JACM), v. 35, n. 2, p. 288-323, 1988.

[Ezhilchelvan et al. 1995] Ezhilchelvan, Paul; Macêdo, Raimundo and Shrivastava, Santosh. Newtop: a fault-tolerant group communication protocol. Proceedings of 15th International Conference on Distributed Computing Systems (ICDCS96), Vancouver, BC, Canada, May 1995.

[Falai e Bondavalli 2005] Falai, Lorenzo; Bondavalli, Andrea. Experimental evaluation of the QoS of failure detectors on wide area network. In: 2005 International Conference on Dependable Systems and Networks (DSN'05). IEEE, 2005. p. 624-633.

[Felber 1998] Felber, P. The corba object group service : A service approach to object groups in CORBA. 179 p. Tese (Doutorado), Département D'Informatique, École Polytechnique Fédérale De Lausanne, 1998.

[Fisher et al. 1985] Fischer, Michael J.; Lynch, Nancy A.; Paterson, Michael S. Impossibility of distributed consensus with one faulty process. Journal of the ACM (JACM), v. 32, n. 2, p. 374-382, 1985.

[Freitas et al. 2023] Freitas, Allan Edgard Silva; Rodrigues, Luiz Antonio; Duarte Jr, Elias Procópio. vCubeChain: Uma Blockchain Permissionada Escalável. In: Anais do XLI Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos. SBC, 2023.

[Gamage et al. 2020] Gamage, H. T. M., Weerasinghe, H., and Dias, N. G. J. A survey on blockchain technology concepts, applications, and issues. SN Computer Science, 1:1–15, 2020.

[Gorender e Macêdo 2002]. Gorender, Sérgio; Macêdo, Raimundo. A dynamically qos adaptable consensus and failure detector. Proceedings of The IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2002 - Fast Abstract Track, pp. B80–B8, Maryland, USA, June 2002. An extended version has been published in the Proceedings of Brazilian Symposium on Computer Networks and Distributed Systems (SBRC2002), Pages 277–292.

[Gorender, Macêdo e Raynal 2005] Gorender, S.; Macêdo, R. J. A.; Raynal, M. A Hybrid and Adaptive Model for Fault-Tolerant Distributed Computing. *Proceedings of IEEE/IFIP Int. Conference on Computer Systemas and Networks (DNS05).p. 412-42.* Yokohama, Japan, June 2005

[Gorender, Macêdo e Raynal 2007] Gorender, S.; Macêdo, R. J. A.; Raynal, M. An adaptive programming model for fault- tolerant distributed computing. IEEE Transactions on Dependable and Secure Computing, v. 4, n. 1, p. 18–31, January 2007.

[Hegering et al. 1998] Hegering, H. G.; Abeck, S.; Neumair, B. Integrated management of networked systems: concepts, architectures, and their operational application. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1998.

[Hellerstein et al. 2004] Hellerstein, Joseph L. et al. Feedback control of computing systems. John Wiley & Sons, 2004.

[Horn 2001] Horn, Paul. Autonomic computing: IBM's perspective on the state of information technology. 2001.

[Hurfin et al. 1999] Hurfin, Michel; Macêdo, R; Raynal, M,; Tronel, F. A general framework to solve agreement problems. In: Proceedings of the 18th IEEE Symposium on Reliable Distributed Systems. IEEE, 1999. p. 56-65.

[Jalote 1994] Jalote, Pankaj. Fault tolerance in distributed systems. New Jersey: Prentice Hall, 1994.

[Karmakar e Gupta 2007] Karmakar, Sushanta; Gupta, Arobinda. Adaptive broadcast by distributed protocol switching. In: Proceedings of the 2007 ACM symposium on Applied computing. 2007.

[Kopetz 1997] Kopetz, Hermann. Real-Time Systems: Design Principles for Distributed Embedded Applications. Springer, 1997.

[Kumar et al. 2017] Kumar, S; Bhargava, B; Macêdo, R. J. de A.; Mani, G. Securing IoT-Based Cyber-Physical Human Systems against Collaborative Attacks. 2017 IEEE International Congress on Internet of Things (ICIOT), Honolulu, HI, USA, 2017, pp. 9-16,

[Lamport e Lynch 1989] Lamport, Leslie; Lynch, Nancy. Distributed computing: Models and methods. In: Formal models and semantics. Elsevier, 1990. p. 1157-1199.

[Lamport et al. 1982] Lamport, L.; Shostak, R.; Pease, M. The byzantine generals problem. ACM Transactions on Programming Languages and Systems, v. 4, n. 3, p. 382–401, July 1982.

[Lima et al., 2016] Caldeira Lima, A; Rocha, F; Volp, M; Verissimo, P. Towards Safe and Secure Autonomous and Cooperative Vehicle Ecosystems. In Proceedings of the Second

ACM Workshop on Cyber-Physical Systems Security and PrivaCy, CPS-SPC@CCS (2016, October).

[Macêdo 2000] Macêdo, Raimundo José de Araújo. Failure detection in asynchronous distributed systems. In: Proceedings of the 2nd Workshop on Tests and Fault-Tolerance. pages 76-81. Curitiba, Paraná, Brazil, July, 2000. Available on: <https://sol.sbc.org.br/index.php/wtf/article/view/23478/23305>.

[Macêdo 2007] Macêdo, Raimundo J. de A. An Integrated Group Communication Infrastructure for Hybrid Real-Time Distributed Systems. Proceedings of the 9th Workshop on Real-Time Systems (WTR 2007), p.81-88. Belém, Brazil, 2007.

[Macêdo 2008a] Macêdo, Raimundo J. de A. Adaptive and Dependable Group Communication. Technical Report 001/2008. Distributed Systems Laboratory, UFBA. January 2008. Available on http://www.lasid.ufba.br.

[Macêdo 2008b] Macêdo, R. J. A. Approaches for adaptive and dependable distributed systems. IFIP Workshop on Dependability of Large-Scale and Dynamic Systems (IFIP'2008). Natal, RN, Brazil, February 2008.

[Macêdo 2012] Macêdo, Raimundo J. de A. A Vision on Autonomic Distributed Systems. Proceedings of II Workshop de Sistemas Distribuídos Autonômicos (WoSiDA 2012), pp 31-35, 2012.

[Macêdo 2013] Macêdo, Raimundo J. de A. Designing self-manageable protocols for dependable distributed systems: an experience report. Workshop on Dependability and Fault Tolerance. 64rd Meeting of IFIP Working Group 10.4 - Dependable Computing and Fault Tolerance. Visegrád, Hungary, 2013.

[Macêdo e Farines 2018] Macêdo, Raimundo; Farines, Jean-Marie. Projeto de Sistemas Distribuídos e de Tempo Real para Automação. EDUFBA, 2018. 250 p. ISBN 978-85-232-1675-7.

[Macêdo e Freitas 2009] Macêdo, Raimundo José de Araújo; Freitas, Allan Edgard Silva. A generic group communication approach for hybrid distributed systems. In: Distributed Applications and Interoperable Systems: 9th IFIP WG 6.1 International Conference, DAIS 2009, Lisbon, Portugal, June 9-11, 2009. Proceedings 9. Springer Berlin Heidelberg, 2009.

[Macêdo e Freitas 2010] Macêdo, Raimundo José de Araújo; Freitas, Allan Edgard Silva. Group Communication for Self-Aware Distributed Systems. In: Simposio Brasileiro de Redes de Computadores e Sistemas Distribuidos, 2010, Gramado. Anais do Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuidos. Porto Alegre-RS: SBC, 2010.

[Macêdo e Gorender 2008] Macêdo, R. J. de A.; Gorender, Sérgio. Detectores Perfeitos em Sistemas Distribuídos Não Síncronos. In: IX Workshop de Teste e Tolerância a Falhas (WTF 2007), 2008, Rio de Janeiro. Anais do IX Workshop de Teste e Tolerância a Falhas (WTF 2007). Rio de Janeiro: SBC - Sociedade Brasileira de Computação, 2008. v. 1. p. 127-140.

[Macêdo e Gorender 2009] Macêdo, R. J. de A.; Gorender, Sérgio. Perfect Failure Detection in the Partitioned Synchronous Distributed System Model. In: Fourth International Conference on Availability, Reliability and Security (ARES 2009 The International Dependability Conference), 2009, Fukuoka. ARES/CISIS 2009 proceedings. Japan: IEEE Computer Society, 2009.

[Macêdo e Gorender 2012] Macêdo, R. J. de A.; Gorender, Sérgio. Exploiting Partitioned Synchrony to Implement Accurate Failure Detectors. International Journal of Critical Computer-Based Systems (IJCCBS), v. 3, p. 168-186, 2012.

[Macêdo et al. 1993] Macêdo, R. J. de A.; Ezhilchelvan, P; Shrivastava, P., K. Modelling Group Communication using Causal Blocks", 5th European Workshop on Dependable Computing, Lisbon, Feb., 1993.

[Macêdo et al. 2004] Macêdo, R.; Lima, G; Barreto, L; Andrade, A.; Sá, A; Barboza, F; Albuquerque, R.; Andrade, S. Tratando a previsibilidade em sistemas de tempo-real distribuidos: especificação, linguagens, middleware e mecanismos básicos. Capítulo do Livro de mini-cursos do 22o. Simpósio Brasileiro de Redes de Computadores, SBRC'2004.pp. 105-163. Gramado, RS, maio 2004.

[Macêdo et al. 2013] Macêdo, Raimundo José de Araújo; Freitas, Allan Edgard Silva; de Sá, Alírio Santos. Enhancing group communication with self-manageable behavior. Journal of Parallel and Distributed Computing, v. 73, n. 4, p. 420-433, 2013.

[Macêdo, Gorender e Cunha, 2005] Macêdo, Raimundo; Gorender, Sérgio.; Cunha, Paulo. The Implementation of a Distributed System Model for Fault Tolerance with QoS. *In Anais do Simposio Brasileiro de Redes de Computadores (SBRC 2005). p. 827-840,* Fortaleza, May 2005.

[Mills et al. 2004] Mills, K.; Rose, S.; Quirolgico, S.; Britton, M.; Tan, C. An autonomic failure-detection algorithm. In: Proceedings of the 4th International Workshop on Software and Performance. New York, NY, USA: ACM, 2004.

[Milojicic et al. 2002] Milojicic, Dejan S. et al. Peer-to-peer computing. 2002.

[Nakamoto 2008] Nakamoto, S. . Bitcoin: A peer-to-peer electronic cash system. 2008.

[Ogata 1995] Ogata, Katsuhiko et al. Modern control engineering. Upper Saddle River, NJ: Prentice hall, 1995.

[Ongaro e Ousterhout 2014] Ongaro, D. e Ousterhout, J. In search of an understandable consensus algo- rithm. In Proc. of the 2014 USENIX Conference, USENIX ATC'14, USA. USENIX. 2014.

[Poon e Dryja, 2015] Poon, Joseph; Dryja, Thaddeus. The bitcoin lightning network. Scalable o-chain instant payments, 2015.

[Rimal et al. 2009] Rimal, Bhaskar Prasad; Choi, Eunmi; Lumb, Ian. A taxonomy and survey of cloud computing systems. In: 2009 Fifth international joint conference on INC, IMS and IDC. Ieee, 2009. p. 44-51.

[Ruoso et al. 2014] Ruoso, V., Bona, L., and Duarte Jr, E. P. Uma estratégia de testes logarítmica para o algoritmo hi-adsd. InWorkshop de Testes e Tolerância a Falhas, pages 31–44. SBC. 2014.

[Rütti et al. 2006] Rütti, Olivier; Wojciechowski, Paweł T.; Schiper, André. Service interface: a new abstraction for implementing and composing protocols. In: Proceedings of the 2006 ACM symposium on Applied computing. 2006. p. 691-696.

[Sá e Gorender 2019] Sá, Margarete e Gorender, Sérgio. Quality of Context for VANETs: QoC Metrics for Connectivity in VANETs. Proceedings of the 18th International Conference on Ad Hoc Networks and Wireless (AdHoc-Now 2019), Springer, pp 420-431, 2019.

[Sá e Gorender 2021] Sá, Margarete e Gorender, Sérgio. Uma análise da Confiança na Qualidade do Contexto em VANETs. Anais do XI Simpósio Brasileiro de Engenharia de Sistemas Computacionais (SBESC 2021), 2021.

[Satzger et al. 2007] Satzger, Benjamin et al. A new adaptive accrual failure detector for dependable distributed systems. In: Proceedings of the 2007 ACM symposium on Applied computing. 2007. p. 551-555.

[Schilit e Theimer 1994] Schilit, B. N. and Theimer, M. M. Disseminating active map information to mobile hosts. IEEE Network, vol 8 no. 5, pp 22-32, 1994.

[Schneider 1990] Schneider, F. B. Implementing fault-tolerant services using the state machine approach: a tu- torial. ACM Computing Surveys, ACM, New York, NY, USA, v. 22, n. 4, p. 299–319, December 1990. ISSN 0360-0300.

[Shoker e Verissimo, 2022] Shoker, A; Verissimo, P. "Intrusion Resilience Systems for Modern Vehicles - Position Paper". In the 7th *Critical Automotive applications: Robustness & Safety workshop*, CARS@EDCC, September 2022. (See significantly extended version in: Shoker, Rahli, Decouchant, and Esteves-Verissimo. "Intrusion Resilience Systems for Modern Vehicles". In *the 97th IEEE Vehicular Technology Conference: VTC2023,*Florence, Italy, June 2023.)

[Silva et al. 2023] Silva, Francisco A. et al. Avaliação de Desempenho de Blockchains Permissionadas Hyperledger Orientada ao Planejamento de Capacidade de Recursos Computacionais. In: Anais do XLI Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos. SBC, 2023.

[Sivasubramanian et al. 2005] Sivasubramanian, Swaminathan et al. GlobeDB: Autonomic data replication for web applications. In: Proceedings of the 14th international conference on World Wide Web. 2005. p. 33-42.

[Vahdat-Nejad et al. 2016] Vahdat-Nejad, H., Ramazani, A., Mohammadi, T. and Mansoor, W. “A survey on context-aware vehicular network applications”, Vehicular Communications, vol. 3, pp. 43–57, 2016.

[Veríssimo 2006] Veríssimo, Paulo E. Travelling through wormholes: a new look at distributed systems models. ACM SIGACT News, v. 37, n. 1, p. 66-81, 2006.

[Veríssimo e Casimiro 2002] Veríssimo, Paulo; Casimiro, António. The timely computing base model and architecture. IEEE Transactions on Computers, v. 51, n. 8, p. 916-930, 2002.

[Veríssimo e Rodrigues 2000] Veríssimo, P.; Rodrigues, L. Distributed systems for systems architects. USA: Kluwer Academic Publishers, 2000.

[Verissimo et al. 2009] - Verissimo, P.; Correia, M.; Neves, N.; Sousa, P. “Intrusion-Resilient Middleware Design and Validation”, in *Information Assurance, Security and Privacy Services*, ser. Handbooks in Information Systems. Emerald Group Publishing Limited, May 2009, vol. 4, pp. 615–678.

[Vöelp e Veríssimo 2018] Vöelp, Marcus; Veríssimo, Paulo. Intrusion-tolerant autonomous driving. In: 2018 IEEE 21st International Symposium on Real-Time Distributed Computing (ISORC). IEEE, 2018. p. 130-133.

[Wang et al. 2021] Wang, Yibo et al. iBatch: saving Ethereum fees via secure and cost-effective batching of smart-contract invocations. In: Proceedings of the 29th ACM

Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. 2021.

[Wood 2014] Wood, D. D. Ethereum: A secure decentralised generalised transaction ledger. 2014.

[Yu et al. 2019] - Yu, Jiangshan; Kozhaya, David; Decouchant, Jérémie; Verissimo, Paulo. RepuCoin: Your Reputation is Your Power. In IEEE Transactions on Computers (2019).