MELHORIA NA ARQUITETURA DE COMUNICAÇÃO EM SISTEMA LEGADO DE CONTROLE DE TRÁFEGO AÉREO

Talita Dos Santos Suzart
Atech – Negócios em Tecnologia S/A
Rua do Rócio, 313 – Vila Olímpia, São Paulo-SP. (11)3103-4600 2004, tssuzart@atech.com.br

Francisco de Carvalho Naime
Atech – Negócios em Tecnologia S/A
Rua do Rócio, 313 – Vila Olímpia, São Paulo-SP. (11) 3103-4600 2004, fnaime@atech.com.br

RESUMO

Com o crescimento do uso do espaço aéreo, o controle de tráfego aéreo deve ser escalável e adaptável às necessidades e especificidades das região que controla. Devido à criticidade e complexidade do sistema que auxilia o controle de tráfego aéreo, não é possível mudar sua arquitetura de forma abrupta, portanto todas as melhorias na solução devem ser realizadas de forma gradual. Posto isso, a modernização da camada de comunicação é necessária para garantir a troca de mensagens entre os serviços, sendo assim, necessário desenvolver a nova arquitetura de comunicação do sistema legado em uso no Brasil. Essa arquitetura deve ser compatível com componentes externos, aumentar o uso de blocos funcionais e diminuir o acoplamento de dependências de subsistemas.

Palavras-chaves: Protocolo de Comunicação, Sistemas ATC, Sistema Legado, *Middleware*, Blocos Funcionais.

ABSTRACT

With increasing use of airspace, traffic control systems must be scalable and adaptable to the needs and specificities of the region it controls. Due to be a critical and complex system that supports the air traffic controller, it is not possible to completely change its architecture, so any improvement should be gradual. The modernization of the communication layer is necessary to guarantee the messages exchange between services, in order to elaborate the new communication architecture of the legacy system in use in Brazil, this proposal is compatible with external components, increases the use of building blocks and decreases the coupling and dependences of the subsystems.

Keywords: Network Communication Protocol, ATC Systems, Legacy System, Middleware, Building Blocks.

1. INTRODUÇÃO

A complexidade no uso do espaço aéreo tem aumentado gradativamente com a chegada de novas tecnologias (NETO, 2017) e os sistemas de controle de espaço aéreo precisam, cada vez mais, manter a disponibilidade e confiabilidade dos dados e informações utilizadas para as tomadas de decisão. A camada de comunicação deve ser robusta para suportar o volume de dados, problemas físicos da rede e acréscimos de novos componentes, sejam do próprio software ou de outros sistemas integrados.

As premissas para este estudo visam diminuir o acoplamento entre componentes, facilitar a configuração do sistema a partir da instanciação facultativa de componentes, aumentar confiabilidade e a escalabilidade dos requisitos de capacidade do sistema e facilitar a integração com sistemas externos.

O artigo organiza-se em 5 seções, a seção 2 apresenta a metodologia aplicada aos estudos, a seção 3 apresenta a escolha das ferramentas para a elaboração prática, que é apresentada na seção 4 junto aos resultados das provas de conceito e, finalmente, a seção 5 apresenta a conclusão e a sugestão da nova arquitetura.

2. METODOLOGIA

Os métodos utilizados para o estudo foram baseados em aspectos da metodologia de pesquisa comparativa. Os passos do processo classificatório foram divididos da seguinte forma:

• Pesquisa técnica de protocolo de comunicação: inicialmente, algumas ferramentas foram pesquisadas para definir quais se adequariam ao sistema de mensageria, assim aprofundou-se teoricamente em cada uma delas;

• Pesquisa sobre as perspectivas de capacidade de sistemas ATC: então, as necessidades futuras, dada a complexidade esperada, foram levantadas a fim de listar critérios para guiar a próxima etapa;

• Avaliação de ferramentas, para essa etapa considerou-se as perspectivas de capacidades levantadas anteriormente, com a finalidade de realizar uma classificação entre as ferramentas e, assim determinar quais ferramentas seguiriam para a próxima etapa; e

• Execução de provas de conceito: para validar as hipóteses levantadas na classificação e na pesquisa teórica, executou-se testes que visavam a facilidade de uso e performance, inclusive em situação de degradação nas duas ferramentas classificadas.

3. AVALIAÇÃO TÉCNICA DAS FERRAMENTAS

No total, cinco ferramentas foram estudadas e comparadas teoricamente, tendo como requisitos de avaliação as perspectivas de capacidade de sistemas ATC. Para cada um desses requisitos, adotou-se pesos de acordo com sua importância, sendo:

- Desejável: Característica que é muito bom ter, mas não é essencial.
- Importante: Característica que aumenta a eficiência.
- Fundamental: Característica que é necessária, mas não exclui a ferramenta que tiver outras alternativas.
- Mandatório: Característica que exclui a ferramenta caso esta não atenda o requisito.

A seguir, um breve resumo de cada uma das ferramentas estudadas:

O ZeroMQ <http://zeromq.org/> é uma biblioteca *open source* (LGPL) de *middleware* orientada à mensagem. Além de ser multiplataforma e multilinguagem, possui muitos padrões de conexão, como: *Request-Reply, Publish-Subscribe* e *Pipeline*, que o torna versátil. Dentre suas características, ressaltam-se alto *throughput* e baixa latência, além da leveza do código. Sobre seu uso em mercados com características de dados semelhante aos sistemas ATC: em artigo (Dworak, 2011), engenheiros do CERN mostraram o resultado de comparação de diversas ferramentas com o CORBA, utilizado à época. De acordo com os requisitos definidos neste estudo, a melhor escolha para a substituição foi o ZeroMQ. Além disso, a NASA, *National Aeronautics and Space Administration*, emitiu um documento (NASA,

2014) que descreve a arquitetura de comunicação para o PAS 1.0, *Power, Avionics and Software*, um subsistema do AEMU, *Advanced Extraveicular Mobility Unit*, e para a mensageria, o protocolo ZeroMQ foi escolhido.

RabbitMQ, criado em 2007 <https://www.rabbitmq.com/>, fornece um *broker* de mensagens, uma plataforma comum onde as aplicações podem enviar e receber mensagens. Esse *broker* oferece persistência e confirmação de entrega e envio, além de alta disponibilidade. Em seu *portfolio* não foi encontrada nenhuma aplicação crítica.

MQTT <http://mqtt.org/> é um protocolo maduro, simples, robusto, versátil e leve para o transporte de mensagens através de redes não confiáveis ou com alto tempo de latência. Inicialmente criado pela *IBM* em 1999 para funcionar em ambiente remoto, tornou-se público em 2011 com a doação para *Eclipse M2M WG*. Segue o padrão OASIS MQTT e ISO/IEC 20922:2016. MQTT utiliza *broker*, possui níveis de serviço e foi desenvolvido para o transporte de mensagem com o uso de *publish/subscribe*. Seu uso é ideal para mensagerias mobiles, pois sua largura de banda minimiza o uso de bateria e o tráfego na rede.

DDS, *Data Distribution Service*, é um padrão OMG, *Object Management Group*, que adota o paradigma *publish/subscribe*. As bibliotecas suportam diversas linguagens e sistemas operacionais. Possui muitas possibilidades de configuração com o *set* de *QoS*.

O DDS *OpenSplice* <http://www.prismtech.com/dds-community> foi utilizado internamente na Atech anteriormente e os desenvolvedores envolvidos relataram alguns problemas: dificuldades na detecção de erros, perda de mensagens, complexidade de uso mesmo utilizando poucos recursos, dificuldade de configuração e manutenção do QoS, *Quality of Service.* Os desenvolvedores também apontaram alguns aspectos positivos da solução DDS geral, tais como integrações de diversas máquinas e linguagens, possibilidade de muitas implementações, grande abstração da camada de comunicação.

Apesar da versão RTi <https://www.rti.com/> ser a mais madura e utilizada entre as distribuições DDS, no artigo citado na descrição do ZeroMQ, os autores, deixam claro que a implementação *request-reply* é complexa e possui curva de aprendizado indesejada devido à complexidade de sua API. Em comparação com a versão *OpenSplice,* o DDS RTi destaca-se por lidar melhor com mensagens grandes e altas taxas de transferência. O resultado é pior com mensagens pequenas (Bellavista, 2013). A versão RTi da DDS também se destaca por ser muito utilizada em soluções de sistemas embarcados na área militar e financeira.

Apenas duas foram escolhidas para as provas de conceito. As ferramentas RabbitMQ e DDS OpenSplice foram descartadas para a execução das provas de conceito, pois (i) tiveram notas inferiores às outras e (ii) não tiveram nenhum quesito que se destacasse positivamente em relação às outras. Ressalta-se que a nota do RabbitMQ foi prejudicada devido à falta de conhecimento da ferramenta e dificuldade em encontrar as informações na internet. Apesar do MQTT ter obtido nota muito próxima à do ZeroMQ e DDS RTi, o MQTT destaca-se negativamente nos requisitos classificados como fundamental, como conexão par-a-par e *streaming*, além de utilizar um *broker* para a comunicação, aumentando a complexidade da solução e a latência na troca de informações dentro do sistema.

4. PROVAS DE CONCEITO

A prova de conceito das ferramentas ZeroMQ e DDS RTi foi dividida em duas etapas: a primeira foca na facilidade de uso e a segunda comporta os requisitos de performance e tende a simular problemas passíveis de acontecer em campo.

Os testes iniciais com as ferramentas acentuam a vantagem do ZeroMQ no aprendizado da ferramenta, devido à sua simplicidade e boa documentação, portanto é possível avançar no uso da ferramenta sem grandes degraus de conhecimento. Com a DDS RTi não há facilidades na evolução de conhecimento dentro da ferramenta.

O teste seguinte evidencia a limitação da DDS RTi, que apesar de ter capacidade maior de envio (64KB) quando o *QoS profile* é configurado, apresenta perdas de mensagens utilizando-se tanto *Shared Memory Protocol* quanto UDP. O comportamento de perda de mensagens acentua-se quanto maior é o tamanho da mensagem. Nos testes do ZeroMQ (TCP) não houve nenhum erro de inversão de mensagens, tamanho ou conteúdo.

O monitoramento de *Memory Leak* para envio de mensagens C ou Java se mostrou

satisfatório para as duas ferramentas. Assim como tiveram resultados satisfatórios e similares no envio de mensagens REQ-REP, apesar de terem implementações bem distintas.

Nos testes de recebimento de mensagens de mais de uma origem para mais de um destino, a lib do ZeroMQ não possui meio de identificar a origem de cada mensagem automaticamente, porém a implementação é trival e funciona conforme esperado. Na DDS RTi o *Shared Memory Transport* é utilizado para a comunicação inter processo e uma fila é criada para a troca das mensagens. Para testá-lo configurou-se 2 *publishers* e 2 *subscribers* rodando na mesma máquina, todos utilizando *Shared Memory Transport*. Quando o *sleep* de envio foi configurado para ser igual ao de leitura, houve perda de mensagem em um dos *subscribers* na conexão do 2º *publisher*. Posteriormente, o teste foi realizado com a frequência de envio dez vezes maior que a leitura, porém, após a conexão do segundo *publisher*, apresentou muitas perdas durante a execução. No teste UDP, configurou-se 2 *publishers* em uma máquina virtual e 2 *subscribers* em outra, com frequências de envio e recebimentos diferentes. Durante a execução, os dados não são perdidos, apenas os iniciais e finais devido à desconexão e tamanho do cache.

Um sistema ATC deve continuar a operação mesmo se houver queda ou acréscimo de nós na rede. No ZeroMQ, o tempo para conexão é configurável, identificou-se algumas variações aleatórias nos testes, que são propositais para evitar *reconnection storm*. Nos testes de reconexão no ZeroMQ, verificou-se que a aplicação recebeu todas as mensagens enfileiradas, apesar do ZeroMQ não garantir que o socket irá aceitar a quantidade definida, o limite pode ser 60-70% inferior. A implementação da DDS RTi pode receber dados antigos desde que esteja configurado para tal, essa configuração é dada em níveis e afetam a performance do envio de mensagens.

No teste seguinte, mediu-se o tempo em que um *subscriber* leva para receber a primeira mensagem, incluindo o tempo de carregamento da biblioteca, conexão, *hand-shake* e inicialização. Como esse tempo é proporcional ao tamanho da mensagem, utilizou-se mensagens de 10000 bytes. Os testes foram executados dez vezes e o resultado foi considerado satisfatório, a *tabela 1* traz a comparação desses resultados.

Tabela 1: Tempo de recepção da primeira mensagem

	ZeroMQ	***DDS RTi***
Média	2143 µs	286532 µs
Máxima	2432 µs	302902 µs

No teste de perda de mensagens, observou-se que no ZeroMQ é necessário incluir um número sequencial na mensagem para identificar a perda de mensagem na recepção, porém para identificar a perda no envio é necessário alterar a lib. A DDS RTi pode ser configurado como *reliable* para evitar perdas de mensagens quando essas são enviadas com alta frequência. Algumas mensagens foram perdidas apesar da ferramenta DDS RTi não acusar perdas, só foi possível determinar a perda pelo *serial number* incluído pela aplicação de teste e verificação da mesma no recebimento.

Observou-se que o tamanho da mensagem é limitado pela memória disponível, tanto com o ZeroMQ, quanto com a DDS RTi. Com os testes, conclui-se que o ZeroMQ comporta mensagens muito grandes (testado com mensagens de 100MB), mas, como as filas são dimensionadas por número de mensagens, é necessário configurá-las de acordo com o tamanho das mensagens e a quantidade de memória que se deseja utilizar. Outro ponto a considerar é que, no ZeroMQ, as mensagens são entregues inteiras ou não são entregues. Portanto, aumentar o tamanho das mensagens aumenta a quantidade de memória utilizada. Com uma mensagem grande, a DDS RTi fragmentará e rejuntará o dado automaticamente para acomodá-lo no limite máximo da camada de transporte, no entanto, alterar as configurações do *QoS policies* pode ter um impacto significativo na performance. É necessário que o *dataWriter* esteja configurado para ser *reliable* e assíncrono, redimensionar o *buffer* do *socket* e

definir um controle de fluxo para evitar fragmentação da memória.

Nos testes de performance com o ZeroMQ, nenhuma mensagem foi perdida, a CPU oscilou entre 0 e 4% com média de 1,70% no *subscriber* e entre 0 e 7% com média de 5,02% no *publisher* e não apresentou tendência de alta.

Um sistema ATC possui algumas necessidades de envio de mensagem diferentes, de acordo com as características do uso desses dados. Uma delas é a entrega de mensagens sem garantia de entregas e outros tipos de mensagens precisam ter garantia de entrega. Ao contrário da solução da DDS RTi, o ZeroMQ não atende completamente as duas características, para que o ZeroMQ atenda a entrega sem garantia é necessário habilitar o ZMQ_CONFLATE para desligar as filas de envio e recepção. Dessa forma, há renegociação de pacotes perdidos (TCP), porém não há enfileiramento de mensagens.

Para teste de *throughput*, o *publisher* enviou mensagens sem intervalo entre elas, de modo a garantir que as filas permanecessem cheias. O *subscriber* lê tantas mensagens quanto possível, fazendo o cálculo do *throughput*. Na situação do teste, o DDS, linha verde no *gráfico 1*, ficou extremamente lento com mensagens a partir de 6 KB. Configurou-se, então, um controle de fluxo no *QoS*, o que limitou o *throughput* a aproximadamente 450Mbps. O ZeroMQ, linha vermelha do *gráfico 1*, estabilizou-se em 900Mbps.

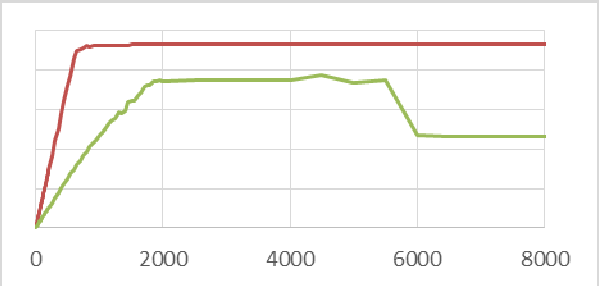


Gráfico 1: Throughput por tamanho de mensagem

No teste de comunicação com rede com restrição de latência, mediu-se o *throughput* para diversos tamanhos de mensagens (5 a 10000 kB) em diversos valores de *delay* (0 a 1000 ms). O *delay* foi criado a partir do tc-netem, em ambos os sentidos.

No ZeroMQ, *gráfico 2*, e na DDS RTi, *gráfico 3*, as curvas de Mbps por tamanho de mensagem, eixo vertical, têm o mesmo padrão entre si para todos os *delays* aplicados. Observa-se que quanto maior o *delay*, maior deve ser o tamanho da mensagem até que o *throughput* se estabilize.

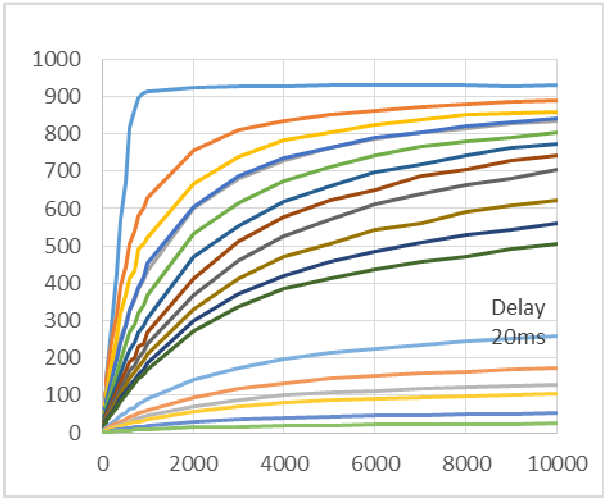


Gráfico 2: Padrão das curvas de Mbps por tamanho de mensagem com restrição de latência ZeroMQ.

No DDS RTi, foi necessário configurar um controle de fluxo específico no *QoS* para que a transmissão não cessasse com o aumento da latência. O controle de fluxo diminui o *throughput* quando não há *delay*.

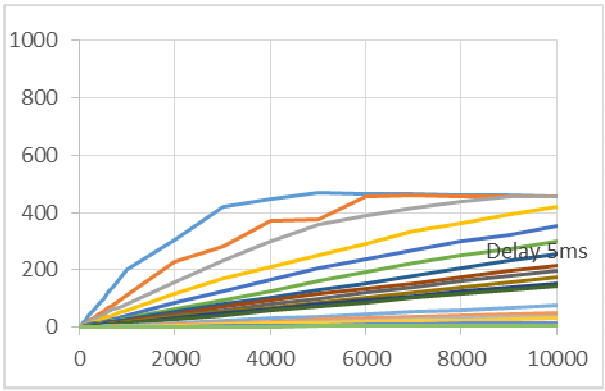


Gráfico 3: Padrão das curvas de Mbps por tamanho de mensagem com restrição de latência DDS RTi.

Para o teste de comunicação com perda de pacotes, mediu-se o *throughput* para diversos tamanhos de mensagens (5 a 10000 kB) em diversos valores de perda de pacotes (0 a 10%). A perda foi criada a partir do *tc-netem*, em ambos os sentidos.

No ZeroMQ, a retransmissão dos pacotes é feita pelo TCP. As curvas de Mbps

por tamanho de mensagem têm o mesmo padrão entre si, *gráfico 4*, para todas perdas aplicadas. Observa-se que a queda do *throughput* é gradual, chegando a 1,3 Mbps com 10% de perda.

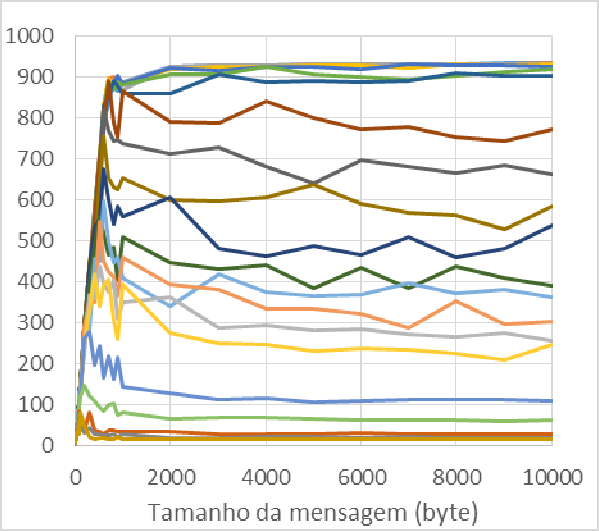


Gráfico 4: Padrão da curva de Mbps por tamanho de mensagem com perda de pacote ZeroMQ.

Para a DDS RTi, foi utilizado o modo *reliable* no *QoS*, para garantir retransmissão dos pacotes perdidos. O DDS RTi apresenta, *gráfico 5*, queda mais abrupta do *throughput* em função do aumento da perda. A partir dos 0,7% a comunicação ficou extremamente lenta e o teste não prosseguiu.

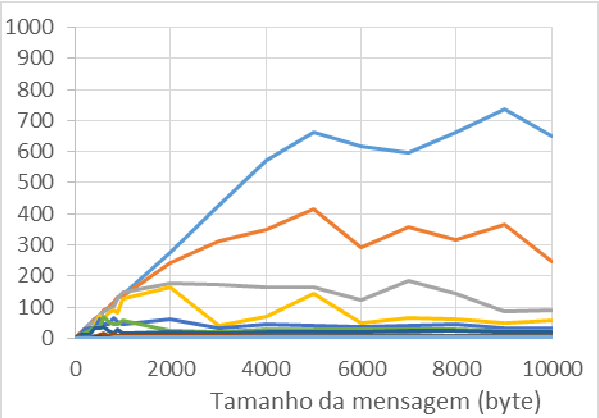


Gráfico 5: Padrão da curva de Mbps por tamanho de mensagem com perda de pacote DDS RTi.

Para teste de comunicação com limitação de banda, utilizou-se a configuração de *QoS* do *switch,* que permite a configuração de *rate* de cada porta em cada sentido, e somente valores múltiplos de 64 kbps. Para os testes configurou-se as portas em ambos os sentidos com os valores: 64kbps, 1024kbps, 9984kbps e 1000032kbps.

No ZeroMQ, *gráfico 6*, quanto menor o *rate*, menor o tamanho da mensagem até que o *throughput* torne-se constante. Para *link* de 100Mbps, mensagens a partir de 70 *bytes* atingem mais de 90% do *rate* do *link*.

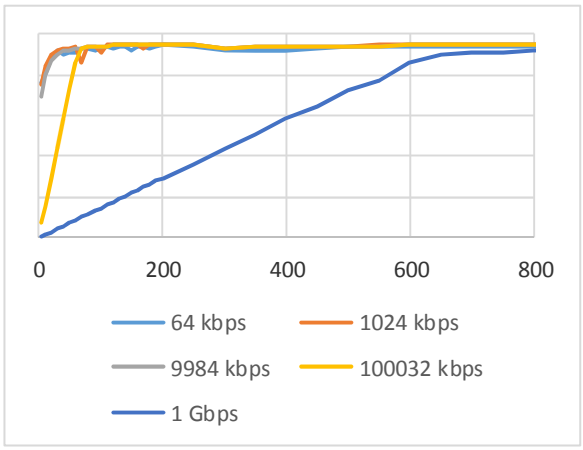


Gráfico 6: Padrão da curva de link (%) por tamanho de mensagem em comunicação com limitação de banda, ZeroMQ.

Com DDS RTi, *gráfico 7,* a porcentagem utilizada do link fica instável. Com *rate* de 64Kbps, não houve transmissão de mensagens maiores que 350 bytes.

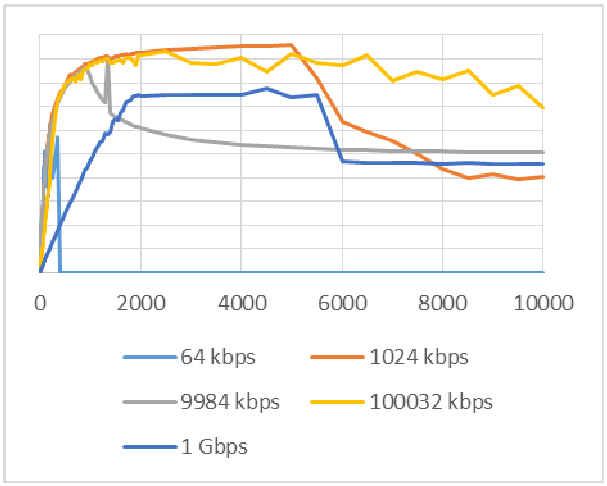


Gráfico 7: Padrão da curva de link (%) por tamanho de mensagem em comunicação com limitação de banda, DDS RTi.

5. CONCLUSÃO

A solução da DDS RTi é bem completa e apresenta níveis de configuração através dos QoS. No entanto, para dominar

essas configurações, há necessidade de conhecimento muito extenso da biblioteca. Em um dos testes executados, percebeu-se que o uso da *shared memory* pelo protocolo não é performático. Muitas vezes há memória livre, mas não é possível utilizá-la. As configurações da DDS RTi, para melhorar a performance, dependem altamente do *Hardware* em que será executado. Além disso, a ferramenta obteve resultados ruins, com muitas perdas de mensagens.

O ZeroMQ é simples e cumpre sua função de protocolo de comunicação. Ele deixa de atender algumas das necessidades levantadas no início do estudo, por exemplo, log de perdas e fila de priorização, sendo que a principal fraqueza do ZeroMQ é que ele não faz envio de mensagem em *broadcast*. Outra fraqueza é a diminuição do *throughput* em redes de maior latência e com perda de pacotes.

Considerando os requisitos levantados inicialmente, a solução da DDS RTi não é adequada ao sistema atual de tráfego aéreo, pois a arquitetura do sistema difere muito do modo de uso da ferramenta.

A alternativa que esse estudo aponta é usar uma solução híbrida, ZeroMQ com UDP. Apesar de não atender alguns dos requisitos levantados inicialmente, eles são facilmente configurados com uma camada acima. Na solução proposta, ter-se-á (i) comunicação TCP, além de UDP, garantindo a entrega e retransmissão de pacote perdido, (ii) tamanho de mensagem não limitado (sobre UDP limitado a 65KB), (iii) ordenação de mensagem e (iv) controle de fluxo e congestão.

6. REFERENCES

A. Dworak, F. Ehm, W. Sliwinski, M. Sobczak, CERN, Geneva, Switzerland, “Middleware Trends and Market Leaders 2011”. Disponível em: <http://go.web.cern.ch/go/G9RC>. Acessado em 01 Fev. 2017.

M. Rostanski, K. Grochla and A. Seman, "Evaluation of highly available and fault-tolerant middleware clustered architectures using RabbitMQ," 2014 Federated Conference on Computer Science and Information Systems, Warsaw, 2014, pp. 879-884.

N. Estrada and H. Astudillo, "Comparing scalability of message queue system: ZeroMQ vs RabbitMQ," 2015 Latin American Computing Conference (CLEI), Arequipa, 2015, pp. 1-6.

NASA, William D. Ivancic, Obed S. Sands, Casey J. Bakula, Daniel R. Oldham, Ted Wright, Martin A. Bradish, Joseph M. Klebau, “Power, Avionics and Software Communication Network Architecture”, 2014. Disponível em: < https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20140012559.pdf>. Acessado em 08 Fev. 2017.

Neto, E. P.; Baum, D. M.; Almeida, J. R. A.; Camargo, J. B. C.; Cugnasca, P. S.; Oliveira, I. R.; Mello, M. P.; Fregnani, J.A. T. G.; “Analysis of the Brazilian Airspace Capacity Model in Complex Environments”, XVI Air Transportation Symposium (SITRAER), Rio de Janeiro, RJ, Brasil, 2017, pp. 625-637.

P. Bellavista, A. Corradi, L. Foschini and A. Pernafini, "Data Distribution Service (DDS): A performance comparison of OpenSplice and RTI implementations," 2013 IEEE Symposium on Computers and Communications (ISCC), Split, 2013, pp. 000377-000383.

V. M. Ionescu, "The analysis of the performance of RabbitMQ and ActiveMQ," 2015 14th RoEduNet International Conference - Networking in Education and Research (RoEduNet NER), Craiova, 2015, pp. 132-137.