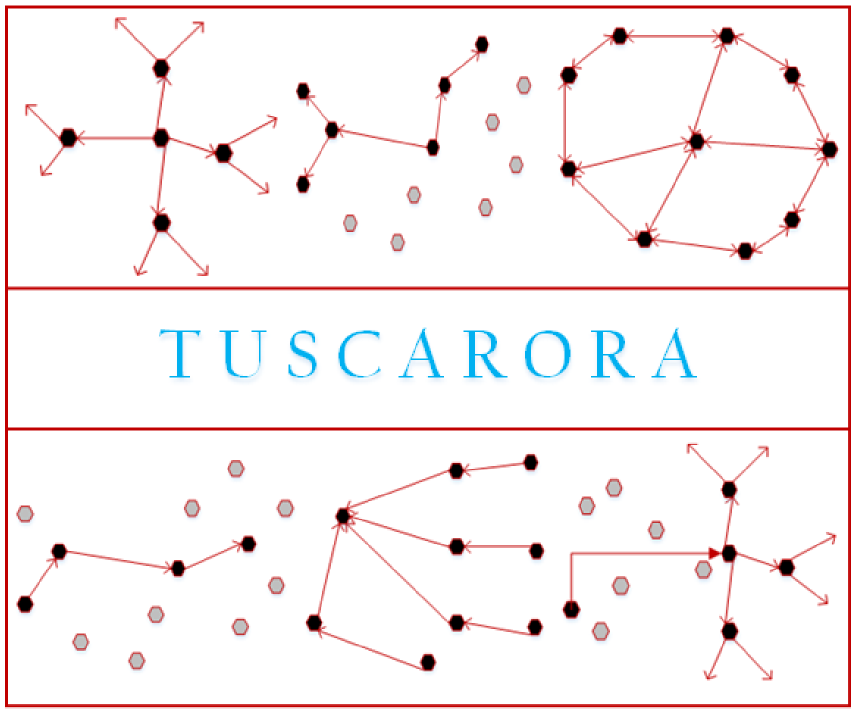


Tuscarora Product Manual

The Samraksh Company

Document Version 1.0

This research was developed with funding from the Defense Advanced Research Projects Agency (DARPA). The views, opinions, and/or findings contained in this article/presentation are those of the author(s)/presenter(s) and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

This page is left blank intentionally.

Executive Summary

In the near future, scaling wireless networks is going to be limited by technology constraints stemming from the architectural decisions made during the early days on the Internet. This does not imply that these were wrong decisions, but the current architecture does impact future growth and evolution. In some domains we are already seized by this problem.

Tuscarora is a breakthrough technology that solves this critical upcoming problem in both military and civilian networking space, by providing a new clean slate networking architecture that allows the building of very large scale, capacity efficient, hardware portable wireless networking systems.

The breakthrough technology consists of multiple software components; (1) a framework, (2) one or more router modules called ASNPs, (3) a policy manager, and (4) reworked applications

Samraksh provides the framework, policy manager and some example ASNPs and Applications. Users are encouraged to develop custom ASNPs and apps. More are expected to be available from the user community in future. The framework, patterns and apps are platform independent, portables and integrated with a network simulator for testing.

The software is designed using modern object oriented programming principles in C++. It is efficient, scalable, portable and supports heterogeneous radios. It does not require the legacy networking stack based on TCP/UDP/IP, but can co-exist with it.

We illustrate a complex military use case of a Futuristic Forward Operating Base, which can be accomplished using Tuscarora/ASNPs, which otherwise would require several times more bandwidth (and hence money) and might still not be satisfactorily implemented.

Samraksh is open sourcing this technology. Samraksh is providing Tuscarora/ASNPs available free of charge for academics and U.S. Government use. Other institutions can choose to license the technology, and/or work with Samraksh to implement prototypes.

This page is left blank intentionally.

Table of Contents

Table of Figures

This page is left blank intentionally.

1 Overview and Context

1.1 Introduction

Welcome to the Tuscarora Product Manual!

The aim of this document is to help a reader understand what Tuscarora is and how to use it. Questions, comments or suggestions regarding this manual can be sent to Tuscarora-support@samraksh.com.

If you're already familiar with Tuscarora, and just need to get started installing, developing or using it, feel free to skip forward to Section 2.

On the other hand, if terms like Application Specific Networking Patterns are new to you, or you are trying to understand if it is worth your time to bother reading the rest of this manual, then you might want to skim the rest of this first section to get oriented.

1.1.1 Punchline

Here's the 30 second reason describing the value proposition of Tuscarora:

The Internet has been one of the most important technologies in the history of humankind, but its' architecture may not scale to meet the evolving needs of the next few years, particularly in the high growth area of wireless Internet (edge) devices.

DARPA, the people who brought us the Internet, has funded the development of an alternative solution, "Tuscarora".

The successful testing and now commercial availability of Tuscarora unlocks new opportunities across the entire spectrum of the Internet of Everything (IoE) space.

1.1.2 Background and problem statement:

January 1, 1983 has become known as the birthdate of the Internet, for it is on this date that ARPANET (developed and funded by DARPA) officially switched over to TCP/IP as its principal protocol. Just over 30 years later, we now have nearly as many devices (6.4 billion) connected to the Internet as there are people on the planet (7.4 billion), and by 2020 Gartner and others forecast there will be over 20 billion devices connected to the Internet and by 2030 over 50 billion devices.

However, to get there, we need to grow beyond TCP/IP and our standard networking topologies. If we don't, the Internet is going to scale to meet the widely anticipated near future use cases.

To understand this, consider for a moment the case of vacuum tubes. Vacuum tubes transformed electronics in the early part of the 20th century, and ability of a vacuum tube to create digital signaling with their on/off states heralded in millions of radios, phones, televisions and primitive electronic computing devices. But vacuum tubes were large, slow, consumed a lot of electricity, and generated a lot of heat. Simply put, they didn't scale into the common electronic use cases we are surrounded with today.

But the transistor, invented in 1947, came along and unlocked a new wave of electronics. Transistors scaled to literally trillions of digital electronics, by providing ever-smaller, cheaper, energy efficient, and more powerful ability to manipulate the flow of electrons, and generate the ones and zeros that so enable the world around us today.

TCP/IP may very well be the vacuum tubes of Internet networking.

Similarly, just like the vacuum tubes did for electronics during WWII, TCP/IP works very well for what we do with it today, particularly since the adoption of IPv6 in 2013. However, Transmission Control Protocol is not likely to work well for what's likely to happen over the next few years.

The primary reason for this is that TCP was designed for a wired world, and establishes an end-to-end socket or connection, and requires very high reliability in the in-between links/hops. When a packet fails in one of the in-between links, TCP attempts to do the following: resend the packet all the way from the end-host to either repair the current socket or to reestablish a different socket.

This combination of TCP/IP works very well for our current Internet, which is predominantly wired. While we all see the proliferation of "wireless" around us today, in reality, most "wireless" networks today are really wired networks, usually with just the last hop or link being wireless.

In this wired world, TCP's simple, efficient, connection oriented service adds reliability to the inherently unreliable IP network protocol, while its flow control allows increased data sending rates until congestion is detected. In fact, it is TCP's reliability that make it the Internet's primary protocol, versus alternatives like UDP which can be faster but are less reliable in providing assured delivery of most of the data that is expected. In a wireline environment, where much of the dropped connections are due to congestion in various links in the connection, this provides more reliable connection with faster throughputs However, this reliability comes at a cost, as TCP has more overhead built in to reliably overcome the ordinary hiccups and potholes in the information super highway.

However, TCP does not translate well to the dynamics of wireless and/or mobile networks where increasingly more of the links are wireless, not just the last hop. Here, in these increasingly common situations links can fail not only because of congestion, but because also because of handoffs as well as Over-the-Air (OTA) issues like failure in the radio links (which is much more likely when one or more devices are mobile).

TCP bogs down in these scenarios.

To compensate to these the limitations of TCP and IP (since they are literally baked into the DNA of our networking), we've made wireless networks as-wired-as-we-can to sustain what we know – TCP. Anything else would have required a "clean slate approach" to networking.

However, in the future of billions or trillions of connected things, we can't simply run wires closer and closer to everything. We have to rethink wireless networking for the future (which is already here).

DARPA, the people who brought us the original TCP/IP (and the mouse, Windows, GPS, Siri, cloud computing and many other seminal platforms of our technology age), were among the first to realize the limitations of TCP.

For the last several years, DARPA has been pioneering work on next generation networking protocols, particularly those appropriate to mission critical wireless communications among large numbers of mobile assets. This Wireless Network after Next (WNaN) project has received hundreds of millions of dollars in investments, but has yielded few solutions that can scale to meet the rigorous challenge of the modern battlefield. Effective solutions require hundreds or thousands of nodes connecting to each other in a highly mobile environment, across a variety of radio platforms, with efficient energy usage and high reliability.

To achieve this, a new clean slate approach was needed – abandoning the legacy tenets of TCP/IP and hierarchical topologies, and so DARPA tackled this challenge by inviting the world's leading technical, business and academic organizations to submit proposals to solve this next generation wireless problem.

Through a competitive selection process, DARPA down-selected to only a few companies with viable candidate solutions, one of which is ASNP/Tuscarora.

This manual documents Application Specific Networking Patterns (ASNP) that DARPA selected, funded and tested. This clean slate approach has been found to be a successful approach that overcomes the networking and scaling issues that limit the future potential of current networking technologies and protocols.

DARPA then funded Samraksh to develop a software platform that allows Samraksh, DARPA and other entities to take advantage of the benefits of ASNP. Samraksh completed that phase of the project, delivering the Tuscarora Framework.

Tuscarora Framework, when combined with ASNP (together referred to simply as "Tuscarora" in this document), form a radio-agnostic routing protocols layer that is an alternative to TCP based networking, and can be implemented seamlessly across both wireless ad-hoc mesh networking topologies and today's standard hierarchical topologies.

1.1.3 The Solution

The solution is a framework that combines a new networking architecture together with a software platform that allows devices to take advantage of the architecture.

To understand this, another analogy is needed. Imagine you have a traditional car, but you want a hybrid. However, you've just learned that there is a revolutionary new way to convert your gasoline powered car into an electric hybrid. The plans keep ALL of your existing car in place, but introduce the new electric generators, motors and battery storage for you to run on electric, and you get to choose when to run on gasoline or electric power.

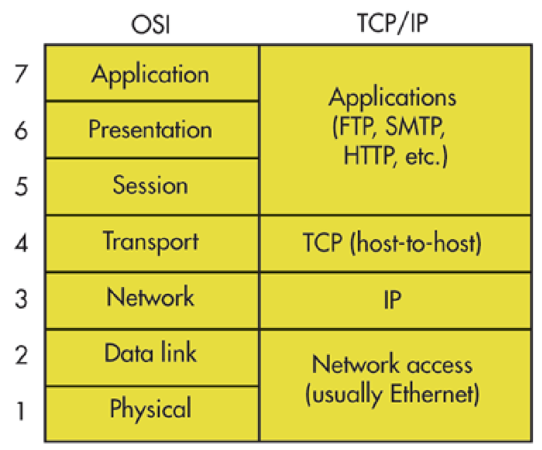


Figure 1. OSI protocol stack vs TCP/IP stack

The inventor of this new technique has made plans for it to be available for free, and if you are so inclined you can make your own parts, buy them from other third parties, or buy them from the inventor.

Let's translate the analogy back to traditional networking. If you are reading this manual, you are very likely to be intimately familiar with the classic software stack of IP. However, for the sake of completeness let's review the protocol stack.

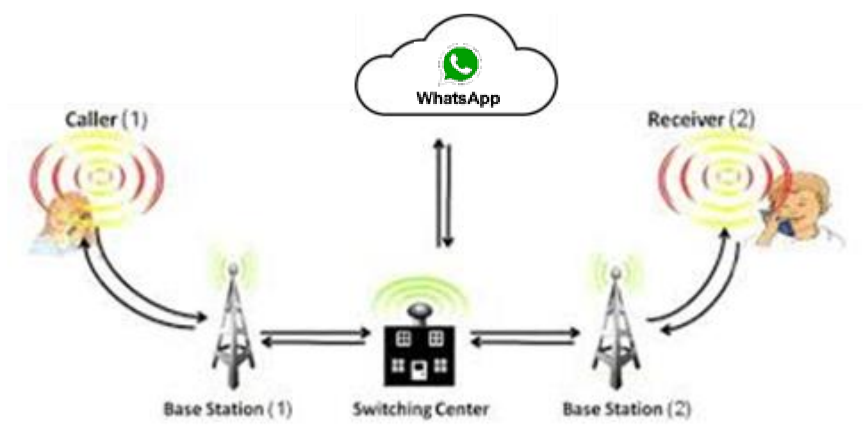


Figure 2. Simplified WhatsApp connection setup

There are two primary models for describing the stack: OSI's 7 layers, and simpler 4 layer TCP/IP stack shown in **Figure 1**. These roughly correspond, and can be used interchangeably to describe an approach to networked applications. Both models are different ways of describing the traditional model of connection that applications use today. In the Figure 1, you can see that OSI's Application, Presentation and Session layers are just more granular levels within the 4 layer TCP/IP application layer. Likewise, the TCP layer is just a specific protocol of the more generic OSI Transport layer, and IP is a specific (and the predominant) version of a Transport layer.

Let's apply those layers to a typical use case today – using your Smart Phone to use WhatsApp with a friend. The call setup will look as shown in **Figure 2**. On Caller 1's end, the first thing we note is that the application is a client, and works in a client-server relationship with the WhatsApp platform (not directly to the software on Receiver (2)'s phone). The local application is a client that connects up the WhatsApp server platform, and that platform connects to the client/application the other phone. Both are on mobile phones, so the underlying network that enables the application client is made via respective cell phone providers. Of course, this is a grossly over-simplified model, that

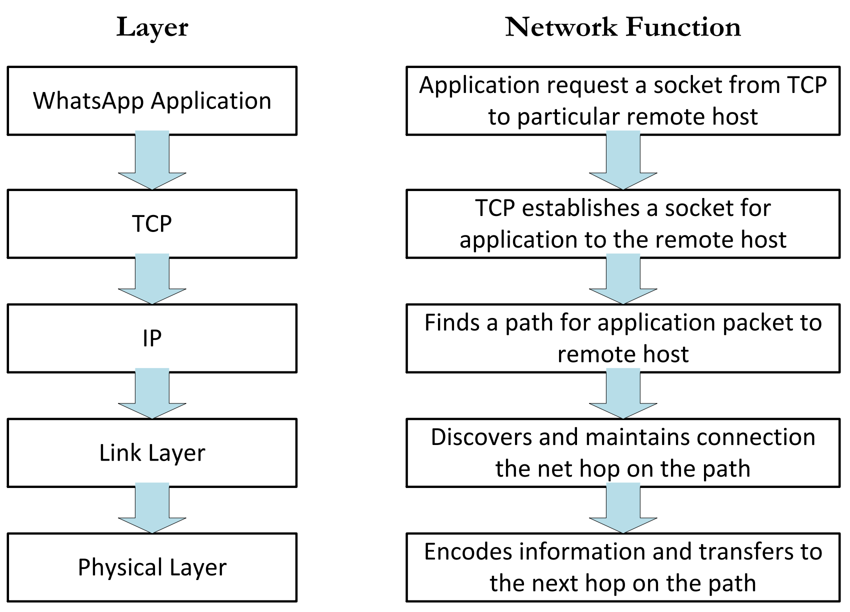

Layer
Network Function
WhatsApp Application
Application request a socket from TCP to particular remote host
TCP
TCP establishes a socket for application to the remote host
IP
Finds a path for application packet to remote host
Link Layer
Discovers and maintains connection the net hop on the path
Physical Layer
Encodes information and transfers to the next hop on the path

Figure 3. Network software stack that enables WhatsApp and their function

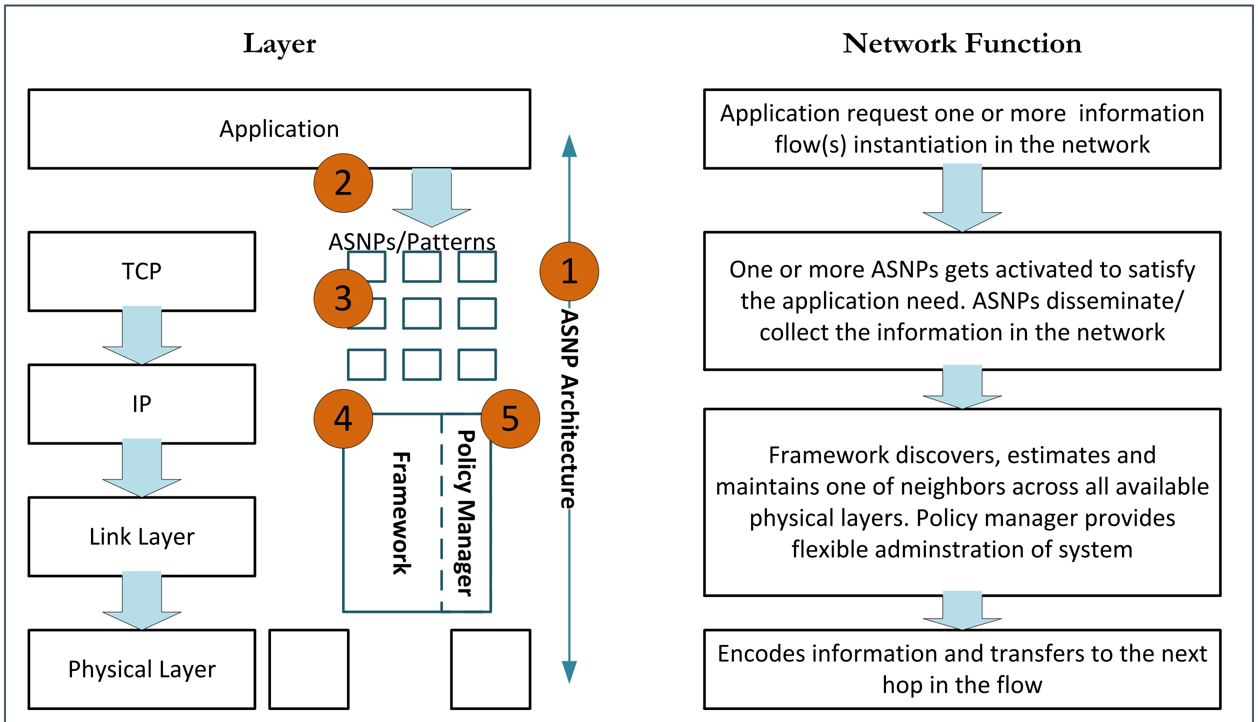

Layer
Network Function
Application
TCP
IP
Link Layer
Physical Layer
ASNPs/Patterns
Framework
Policy Manager
ASNP Architecture
1
2
3
4
5
Application request one or more information flow(s) instantiation in the network
One or more ASNPs gets activated to satisfy the application need. ASNPs disseminate/ collect the information in the network
Framework discovers, estimates and maintains one of neighbors across all available physical layers. Policy manager provides flexible adminstration of system
Encodes information and transfers to the next hop in the flow

Figure 4. Tuscarora stack and its functions

disregards the core transport links between the cell phone companies and the various virtualized physical presences that make up WhatsApp platform, but we are simplifying here without compromising the critical message.

On Caller(1) end of the call, the software stack that enables the WhatsApp experience could be described as shown in **Figure 3.** In our Hybrid car metaphor, this is our gasoline engine.

Now let's look at how Tuscarora creates a parallel networking architecture alongside this TCP, allowing devices to operate in "hybrid" mode. The Tuscarora Application Specific Networking Pattern Architecture, when installed on the mobile phone in this example, would result in a software stack as shown in **Figure 4.**

The graphic in **Figure 4** summarizes the complete Tuscarora Platform, and is analogous to the electric drive system added to a gasoline car system. All of the original gasoline parts are still there and functional, but now there is a separate drive system available to application if they choose to use it (and are configured to use it).

However, it's useful to note that a purely Tuscarora-based model can be built, too. Technology adoption concerns are primarily concerned about how this new technology could be deployed with backwards compatibility into existing platforms. IN the future, however, there may be compelling opportunities to based certain systems on just the ASNP/Tuscarora architecture, with no need for legacy TCP.

Regardless of whether it is implemented in the "hybrid" mode, (alongside legacy TCP applications and architecture, or in a stand-alone "Tuscarora-only" mode, the impacted elements are the same. Those are:

1. The ASNP Architecture. This is the design that enables the platform. It can be described as a set of information flows closely matching the application needs on a peer-to-peer network using almost entirely local information, as opposed to the traditional client-server and hierarchically organized TCP/IP network. See **Section 2.1.2** of this manual for more detail.

2. Application updates that take advantage of the new platform. See **Section 2.2**
3. ASNP Patterns. This is the new software that provides the network information flows. See **Section 2.3** of this document.
4. Tuscarora Platform, that manages the various information flows/patterns and also various radio/transmission technologies. Provides ability to add new radio technologies or instantiate new information flows. See **section 2.4** of this manual for more information.
5. Tuscarora Policies. Provides the ability to setup rules and policies that might be unique to a particular system or organization. Also provides the ability to administer the system at run-time. See **Section 2.5** of this manual for more information.

To facilitate the introduction of these changes to the above elements, Samraksh provides the following resources listed below. Samraksh provides these modules free of charge to academic institutions and the US Federal Government.

1. ASNP Architecture. The architecture and the concept has been published in the public domain, and documented for free public use. ASNP Architecture outline is contained in **Section 2.1** of this document, and published references available at [1]**.**
2. Application developer resources, which are available here: [2]. Application developers will see significant benefit from using the Tuscarora Platform, including additional use cases not possible under TCP, as well as (more). Developers will benefit from orienting themselves to the Tuscarora platform by downloading the latest Tuscarora release package (available at [2]), and reviewing this manual.
3. Samraksh provides a basic set of patterns starting from Tuscarora release version 2.0, and these can be used under license from Samraksh. Developers can also develop their own patterns, and are encouraged to do so. Resources to assist in the open source development of patterns can be found in **Chapter 5** of developers manual available at [2]
4. Samraksh provides the base Tuscarora Software as part of the Tuscarora release packages, and this can be used under license from Samraksh. Tuscarora Software is detailed in **Section 2** of this manual. Note that in a peer-to-peer environment there is no centralized server software, and therefore at a minimum the Tuscarora framework software and the pattern software will have to run on every node in a network.
5. Samraksh provides a basic Tuscarora Policy Management starting with Tuscarora framework version 2.0, and this can be used under license from Samraksh. Tuscarora Policy module is detailed in **Section 2.5** and its design details in **Section 3.12** of this manual.

1.2 Targeted audience and conventions for this manual

This manual should serve as a useful reference to a number of different populations:

- **High Level reader**: Technologists and executives seeking to use ASNPs and Tuscarora to develop compelling new technologies. This manual should provide these readers with a solid introduction to the concepts of ASNP/Tuscarora (I.E. it briefs at a high level on all 5 areas of **Figure 4**, with deeper information on 1,3,4 &5). However, some of the material in this document may go into too much detail for this population, so high level readers will want to read this Introduction section, and skim the Table of Contents to prioritize other areas of interest specific to their needs.

- **Architects and Developers**: Technologists doing "deep dive" due diligence will find this manual to be very useful. It provides high level detail on all 5 areas of **Figure 4**, with mid- and low-level information on 1,3,4,& 5, in sufficient detail necessary to understand and to develop against the whole of the ASNP/Tuscarora Framework, and lists resources for additional resources for more targeted detail beyond the scope of this manual.
- **Application developers:** Technologists seeking to tailor existing end-user applications to the network experience enabled by ASNPs/Tuscarora should find this a useful first step manual. After becoming familiar with the material here, they will want to do a deep dive on the additional develop resources such as the Developers Manual or the API document available at [1].
- **Academics/Researchers**: Academics and researcher should find this manual to be a useful "applied" resource, for understanding the concept and context of ASNPs/Tuscarora. Depending on their research aims and background, researchers will almost certainly want to read the scientific literature on this project, available [1].

1.3 Expectations from this document

By the end of this manual, the reader should be familiar with the following aspects of ASNP/Tuscarora:

1. **The What:** What is Tuscarora, What are ASNPs and what does it do
2. **The How:** How does one get started using ASNP/Tuscarora, including:
   a. Easy simple steps. After reading this document, one should understand roughly how much of effort is required to integrate framework this a particular product or system, including the step-by-step to getting up and running
   b. High-level design: Modularity, with clean separation of function
   c. Software Features: reader should have good understanding of the software elements that need to be modified, and how (with reference examples provided)
3. **The Why:** This document should provide the reader with a good sense of the type of network use cases which would benefit from and ASNP/Tuscarora implementation
4. **The Where** This document should inform the question of the types of resources that are needed to implement and ASNP/Tuscarora implementation and where to get them.
5. **The How long:** This documentation should inform the careful readers (technologists, academics and business executives) to assess at a high level how long an ASNP/Tuscarora implementation will take. However, this is always a challenging task for most networking solutions.

1.4 Rest of the document

The rest of this document is structured as follows:

- **Section 2** provides a working introduction to the Tuscarora Product, including a quick start guide to get started using Tuscarora, the 5 elements of the stack (as shown in **Figure 4**), the framework design goals and a high level design of the framework.
- **Section 3** provides a deeper dive into the core of details of the framework/design.
- **Section 4** provides a case study of how Tuscarora can be used to build and deploy platforms that enable dramatically new and valuable use cases, within highly scalable distributed applications

2 Tuscarora Product

This section will explain the various pieces of software that make the Tuscarora Product Suite.

First, let us briefly review the Application Specific Networking Pattern Architecture based on which the product is built. What follows is a medium-level overview; interested parties and researchers seeking an even more in-depth understanding can look Section 3, and published material available at [1].

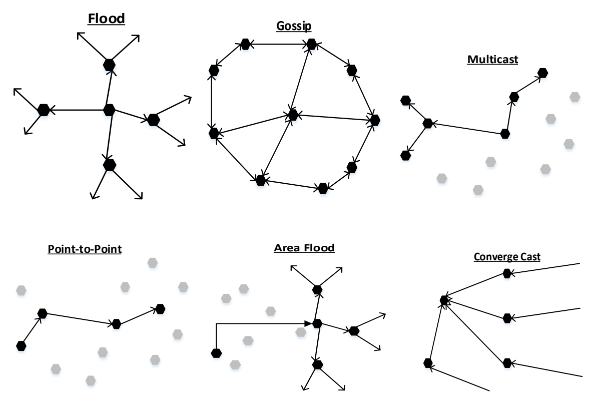


Figure 5. Example information flows used by applications.

2.1 Application Specific Networking Patterns Architecture

2.1.1 Need for a New Architecture

To better understand the architecture, we first need to understand the problem in more detail than was summarized in the Overview section.

The ASNP architecture aims at maximizing the data carrying capacity of wireless networks. Wireless networks are much different than wired networks, like the core internet, in three key ways;

(i) shared spectrum of nodes, a.k.a interference,

(ii) unreliability

(iii) the possibility of mobility and dynamism.

Part of the solution to increase the net available data capacity in wireless networks is spatial reuse. Cellular/mobile phone networks use this technique extensively, albeit in pre-allocated single-hop cells.

It has been known for about 15 years that the aggregate capacity of multihop networks is $\sqrt{n}$ fold greater than an otherwise equivalent cellular network. However, the complexity of multihop network is many times greater if we try to retain the same TCP/IP based internet architecture for wireless networks.

The net effect these factors is that in TCP/IP based multihop wireless networks consume too much control traffic, leaving very little remaining bandwidth for data. And this has led to the whole sale abandonment of the multihop model. Instead, a costly infrastructure based solution has been adopted as the mainstream architecture for deploying mobile networks.

This solution requires treating wireless network as just the last hop of the wired internet, building more and more infrastructure closer to the wireless end points in the form of cellphone towers, WiFi and WiMax access points.

This is the conundrum of wireless networks. A model that offers $\sqrt{n}$ times more network capacity is unused because we are not able to manage the network after many decades of research in the field of "efficient routing". Part of the problem (or even the primary problem) is our reluctance to change the architecture, even though the problem is drastically different.

Samraksh avoids this problem by taking a different approach, starting with a clean slate.

We propose an architecture that can support a network of thousands of nodes in a multi-hop configuration.

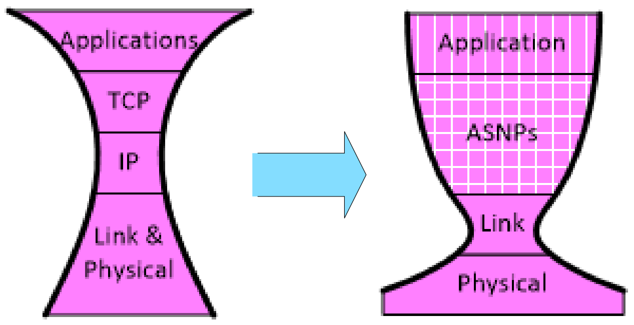


Figure 6. The diversity map of the Internet and the diversity map of ASNP architecture.

The key idea is to throw away the need to support what is called as "routing" between any point in the network to any other point in the network. Instead, the architecture supports multiple simultaneous information flows and map each them to a network pattern. However, it turns out that while the ASNP is a clean-slate design, it can still support IP as needed, as one of the network patterns.

2.1.2 The Architecture: Information Flows

The crux of the ASNP architecture is the idea of identifying one or more "information flows" (for example flows see **Figure 1)** required by the application(s) and pair it with an appropriate network flow. We call such network flow as Application Specific Network Pattern or ASNP (and hence the name of the architecture) or simply "Patterns".

The idea is quite simple and even be argued as intuitive. But its implications are quite radical especially when compared to existing systems:

(i) The architecture must support multiple network patterns simultaneously and therefore for efficiency purposes abstract out services that will be needed by most network patterns.

(ii) The architecture should allow programmability of new network patterns, since it is possible that an application designer can come up with a network pattern that more efficiently implements the information flow needed by the application. This would mean that we will have to discard the single routing paradigm and its de-facto API of point-to-point "sockets" based programming.

2.1.3 Systems Theory and Lowering of the "Waist"

This approach is somewhat equivalent to collapsing the layers or what is called "vertical integration" in systems theory. The general axiom of systems theory is that horizontally integration enhances flexibility while vertical integration enhances performance. While we are interested in increasing performance, a loss of flexibility is not an option. The implication is that complete vertical integration is the anti-model for a system meant for large scale general-purpose deployment. Our solution to this dilemma is two-pronged:

1. To embrace "horizontal integration" by adopting a standardized layer below our patterns layer that provides services required by all patterns (we call this the "neighborhood layer").
2. To allow programmability or addition of new network patterns to the system.

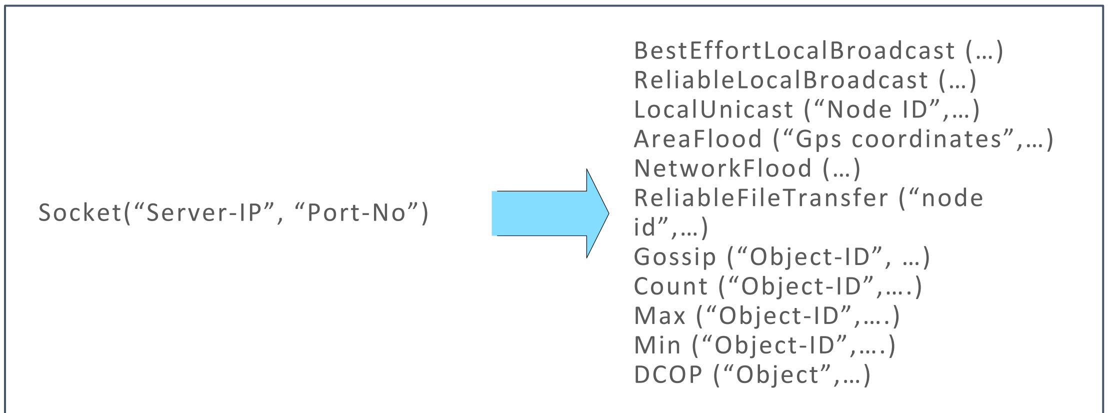
While this might seem like a trivial difference, not all horizontal standards are the same. In fact, the primary adoption of a system depends on getting the layer of standardization right. In general, systems theory proposes [3] that the layer of standardization must be the one with least complexity. For the internet this is the routing layer. We believe that most of the success of the internet stems from getting these integration alignment decisions right. However, for wireless networks, the routing layer is the one of highest dynamism and complexity and hence the wrong point to standardize. The net effect of the ASNP architecture is one of lowering the “waist” (a colloquial word used to refer to the point of standardization) of the system, as shown in **Figure 6**.

To understand the relationship or specificity of patterns to applications, first remember that in TCP you have an infinite:1 ratio – all applications relate to the one networking pattern(TCP).

In ASNP, you have an infinite: infinite relationship, where each Application can invoke (Specifically), the Networking Pattern best suited to its needs.

This supports a very wide and unbounded set of both applications and patterns, but practically speaking, it is likely that a typical instance of the architecture might require only a handful or so very well tested patterns meets the needs of a large number of applications, and each of the patterns having an API that is different and intuitive for the information flow it implements.

2.2 Changes to Application



```
Socket(“Server-IP”, “Port-No”)
```

```
BestEffortLocalBroadcast (…)
ReliableLocalBroadcast (…)
LocalUnicast (“Node ID”,…)
AreaFlood (“Gps coordinates”,…)
NetworkFlood (…)
ReliableFileTransfer (“node
id”,…)
Gossip (“Object-ID”, …)
Count (“Object-ID”,….)
Max (“Object-ID”,….)
Min (“Object-ID”,….)
DCOP (“Object”,…)
```

Figure 7. Sample ASNP APIs in Tuscarora (on right) compared to legacy API (on left)

Certain amount of re-engineering of existing applications will be required when porting them over to the ASNP architecture to make sure of the more efficient network patterns.

This is unavoidable since the ASNPs provide a much richer interface for applications to make exploit. For some examples of APIs that can be provided through ASNPs see **Figure 7.** The scope of these changes varies by implementation, but in many cases this will be a very low level of effort, and in all cases will be dramatically less than the initial level of effort to create the applications in the first place

Sockets --because of their point to point semantics-- forces the network stack to provide arbitrary point to point routing service, which is part of the limitations of the current systems. And because Tuscarora provides this rich set of ASNPs APIs, applications running on Tuscarora can avoids the use of legacy socket and scale to a much larger size.

This richness of APIs would require the previously mentioned level of effort to modify certain components, but the reward is greatly increased scale and efficiency, and enablement of new use cases.

2.3 Application Specific Networking Patterns

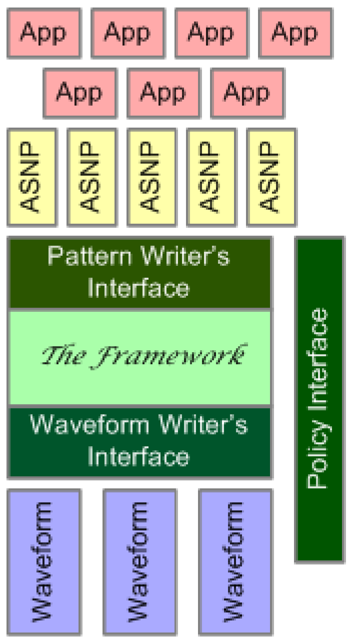


Figure 8. Framework architecture.

ASNPs are the brain of the architecture that actually enable the higher efficiency and an improved user APIs for the applications. In distributed systems, the amount of non-local knowledge a network pattern collects and uses is inversely proportional – the proportionality is sometime linear, sometimes $\mathbf{n^{3/2}}$ or even $\mathbf{n^2}$ depending on the algorithm – to the size of the network it can support. We advocate designing network patterns that only require local network knowledge, resulting in more scalable network patterns. The result is not absolutely scalable (as there is new evidence from P.R. Kumar that even keeping track of your immediate neighbors is not absolutely scalable in the asymptotic sense), but should allow scaling to 10's of thousands of nodes. However, many highly scalable Application Specific Networking Patterns (ASNPs) are known to exist; the problem is none of them are fully general. Hence we allow for multiple of them to coexist. The more scalable a pattern is the more data the pattern can support.

2.4 Framework Software

2.4.1 Design Goals

Figure 8 shows the high-level design of the architecture. If the ASNPs are the brains of the platform, then the heart of the architecture is the framework. The framework is designed to support many simultaneous instances of information flows called as 'Application-Specific Networking Patterns' (ASNPs). The core service provided by the framework is a Standardized Neighborhood Abstraction as part of namespace called the "Pattern Writer's Interface". Similarly, the framework defines a Waveform Writers Interface, which a radio/waveform needs to implement to be compatible with the framework.

Tuscarora software framework is designed with the following goals:

1. **Mix and match between ASNPs and Waveforms:** To act as a layer of standardization for implementing Application Specific Network Patterns. An ASNP once implemented and tested with against the framework should work with any new radio/waveform technology compatible with the framework. Similarly, any waveform that implements the lower interface and is compatible with the framework should support any Pattern that is compatible the framework.
2. **Standardized Neighborhood:** The framework should provide to the Patterns a neighborhood services that estimates the link between nodes and provides metrics for the link in a uniform radio agnostic way. The framework will also provide notification events to the pattern if significant events happen in the neighborhood.
3. **Customizable:** The framework should be customizable depending on the requirements of the application, the software environment and the target platform. Services that are needed must by instantiated, but if already available in either the waveform or the platform they must reused. To this

end we define a number of "External Services" APIs that can be easily wrapped around existing platform services.

4. **Support flexible policies:** The framework should provide an interface for specifying policies by a network administrator. The policy interface need not be standardized and must be extensible and flexible.
5. **Flexible instantiation:** Framework should support flexible instantiation of ASNPs, applications and if possible of Waveforms. Depending on the target platform, if dynamic instantiation is possible it must be supported. Furthermore, it must support multiple instances of the same ASNP type or Waveform type.
6. **Portability:** The framework design should explicitly consider deployment of multiple diverse platforms, such as from bare metal embedded systems to high end multi-SoC/ multi-Core systems. Further, the design should aim to provide zero touch portability of the ASNPs and applications, as long as they use only the PAL API and support libraries that are provided along with the framework

2.5 Policy Manager

The need for network management is integral to any network. For a framework like Tuscarora the need is even more acute, for the framework allows modules and protocols from a number of parties such as application designers, pattern designers, other modules such as link estimation, discovery, time sync and other platform services. Second, the framework's role is also one of resource allocation where the resource is network bandwidth across one or more waveforms. And as far any resource management system, there is a need to set policies of use on the resources for various users. Further, it enables a human-in-the-loop management architecture to carry out network and mission management tasks.

Policy interfaces needs to be much more flexible for 2 reasons; one a number of waveforms might have their own configurations that need to be set. This could also be true for some modules such as link estimation and discovery. Two the set of policies needed could be different between different networks, an Internet of Things (IoT) will need very different policies from say a mobile network or a military network. Adding all of this to a horizontal interface such as the waveform interface would make the interface bloated, confusing and less useful. The solution is to make the policy manager flexible and extensible. **Figure 8** shows policy manager as vertical and as outside of the main framework. This is to suggest that the policy interface is not "standardized" and that it cuts across layers. All though the policy interface is shown outside of the framework in the graphic it is in fact a core part of the framework implementation.

Flexible Management: Tuscarora's design envisions a flexible extensive policy manager. We envision a self-describing interface described using XML or other equivalent languages. Components, especially waveforms, would expose descriptions of their special features (or other control knobs). At load time, the system would search for and connect any mutually understood interfaces.

Current Status: Tuscarora as of version 2.0 supports a policy manager that is not extensible or self-describing (it is the regular compile time based). However, it does support many useful policies and functional policies. The policy manager is expected to be an area of active development going forward.

2.6 ASNPs

As explained in **Section 2.3**, ASNPs and the ability to employ multiple of them to satisfy the networking needs of applications are the core features of Tuscarora that bring the efficiency and scalability.

Consequently, ASNPs are critical pieces of software that needs to be carefully designed and tested. Samraksh provides a few well tested ASNPs, listed below, that are supported out of the box.

Currently supported ASNPs:

- Gossip: A pattern to keep a particular object (software or data) update to date in the network
- Flooding with Pruning: Flood a message to every node in the network that is qualified by a criterion, such as, number of hopes from an originator, or an area bounded by GPS coordinates.
- COP: Common operating picture Pattern provides a common snapshot of every node's state at every node (Under testing)
- Command and Response. Issues a command to all nodes in the network and get a response to that command from every node back to the command originator. (Under testing)

Apart from this list there are a large number of ASNPs that are under various stages of development by Samraksh.

If you are interested in a particular ASNP but do not find it in the current list, please contact Samraksh. Also a large of ASNPS are expected to be available from the community in future.

Finally, Samraksh is willing to provide the support and/or resources for organizations and developers to create their own ASNPs.

2.7 Waveform Shims

Any networked application needs at least one network interface card /radio module to communicate with other nodes. We use the term "Waveform" to generalize radio modules, network interface cards, software defined radios.

The functions that are subsumed by Waveform are coding, modulation, medium access control, etc. That is all functionality below the what is called the Link Layer. Tuscarora is designed to work with any radio/waveform and provide a uniform network experience on top of one or more radio modules, and thus does not specify a given waveform functionality.

To do this the product suite comes with what are known as "Waveform Shims" that enable Tuscarora to work with the various radio modules. Simply put, Waveform Shim is an adaptor layer, that talks the native radio language on the south interface and talks the Tuscarora's Waveform Interface on its north side. Currently, Samraksh provides Shims that work with following radio/waveform modules:

- IEEE 802.15.4
- Wifi
- Ethernet
- Sockets (socket is not technically a waveform, but many NIC just provide a data socket)

The code that Samraksh provides for the Waveform Shims can be quickly adapted to work with most waveform.

2.8 Getting Started with Tuscarora

Tuscarora is a framework that implements the ASNP architecture and supports the deployment of network patterns. Below are steps to start using Tuscarora.

2.8.1 Requirements to start development using Tuscarora

Tuscarora is an object oriented framework written in C++. A developer would need the following to start developing distributed applications and patterns using Tuscarora.

1) A modern host desktop platform such as Linux or Windows that supports the CMake [4] build system
2) A GCC compiler for your target platform that supports C++11
3) System details for all target devices:
    a) Version of GCC supported
    b) Amount of RAM available
    c) CPU architecture
    d) Location to deploy Tuscarora if a file system is available

2.8.2 Steps involved in development

Follow the steps below to get started with Application or ASNP development.

1. Download the Tuscarora source from the repository
    a. Optional: Setup ns3-dce [5] simulator to test code in simulator before deployment
2. Deployment architecture: Decide the deployment architecture, based on the needs of application and target platform capabilities. An application or pattern can either be deployed built with framework as tightly coupled binary, as a dll/shared library or as a separate binary running in its own process
3. Configure CMake files for cross compiling to the target platform under the toolchains directory
    a. Setup the GCC toolchain and the CMake toolchain file under the TuscaroraFW/Platform directory
    b. Modify the main CMakeList.txt file to add the new platform (Several examples are available in CMakeList.txt)
    c. Setup the CMake target for the platform based on deployment architecture as explained in Step 2.
4. Find useful APIs
    a. For application development. Look at **Developer's Manual Chapter 5** to find out-of-box supported ASNP Pattern APIs or develop your own Patterns
    b. For Pattern development. Look at **Developer's Manual Chapter 6** to find Framework APIs.
5. Implement a test for your Pattern/Application
6. Configure waveforms: Tuscarora framework needs at least one radio/waveform to be functional. Each radio/waveform needs a thin shim layer to work with the framework. Write a waveform shim layer or use one of existing shim layers available in Tuscarora source tree. Check Chapter 2 on Developers Manual to run default Waveforms inside ns3 simulator.
7. Configure the Platform using Platform.h file. Check **Chapter 3** on Developers Manual for details.
8. Compile and run tests for target using ./runOrDebug.sh or using mechanism suitable for the target

2.8.3 Quick State Guide

Tuscarora: Quick Start Guide

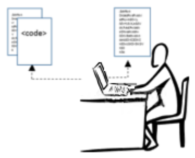
Step 1: Download

Directions

- Go to https://github.com/Samraksh/Tuscarora/releases
- Download latest release
- Follow instructions to installation at https://github.com/Samraksh/Tuscarora/wiki/GettingStarted

Notes

Step 2: Configure and Develop



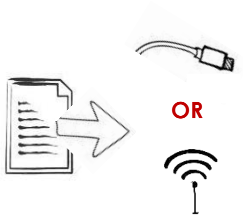
Directions

- Usually some configuration and/or development might be involved
- Users need
  - Tuscarora framework SW (system SW)
  - Patterns (ASNPs)
  - Apps
- Certain pre-configured solutions (containing both a pattern and an app) are already available to run/test. E.G.:
  - Flood
  - Gossip

Notes

- Tuscarora represents a networking platform solution, not an application solution. Therefore providing a broad set of applications out of the box to a broad set of needs is beyond the scope of this release.
- Samraksh envisions that applications will be developed by end-users and an open source ecosystem based on the Tuscarora framework, and would be available progressively following this release.
- Details on how to build Apps are found in Chapter 4 and how to build Patterns are in Chapter 3 of developer's manual

Step 3: Transfer



Directions

- Tuscarora system SW, apps, and patterns can be connected to supported devices via either wired or IP based wireless connections, such as WiFi & Cellular
- For Linux based devices, current version allows user to load Binaries to devices via a experimental Remote orchestrator tool. For details see Chapter 7 of developers manual
- In general, the process is as simple as connecting the cable, copy file to device, then you can execute the file from the command line and/or restart the device

Notes

- This quickstart assumes you have all of the proper authentications
- Devices currently supported include:
  - Linux desktop simulator
  - Native Linux
  - Raspberry Pi
  - Android
- Device support for the following is in development
  - .Net Microframework
- Installation and boot varies by target platform. For more, see Chapter 7 of developer's manual

Step 4: Use

Directions

- Tuscarora is meant to run as a system service on all devices expected to connect via the mesh network
- Tuscarora is envisioned as more of a peer-to-peer network, rather than as a client server network; as such there is no central "server" SW, just the Tuscarora system SW, ASNPs and Apps on the devices on the mesh.
- To run first time, execute appropriate binary
- After initial install, Tuscarora will boot up automatically every time device powers up

Notes

- Application developers are encouraged to think first and foremost about the information flow required by the solution
- An ASNP is the "implementation" of the information flow needed by your application, given the possibilities across the universe of potential links and meshed devices
- Once an ASNP is created, then the developer should write the solution as an App that uses the ASNP (these can be co-designed, as well)

3 Tuscarora High Level Design

3.1 Design Principles

As we dive deeper into the technical details of Tuscarora, it is important to understand its' design principles:

1. **Object Oriented Design:** The design employs some of the classis OO design principles, such as the Single Responsibility, Open Closed, Liskov Substitution, Interface Segregation and Dependency Inversion. Explaining these design principles is outside the scope of this document. Please see Invalid source specified. for more details on object oriented design.
2. **Platform agnosticism and portable:** This is a design requirement to make the core modules platform independent. This requirement influences many design choices throughout the framework and its ancillary modules. Consequently, all the low level modules that interact directly with the OS/Hardware/Platform have been abstracted out as Platform Abstraction Layer.
3. **Event based execution** Invalid source specified.: The framework has been written for an event/interrupt driven design. In event based programming, a program can be thought of as a set of handlers one for each time of event that it is meant to handle. An external program, kernel, hardware or an internal module can be the source of the events, which need to be handled. Event driven programming is in general more efficient than a polling-based multi-thread execution in terms of CPU utilization.
4. **Dependency Injection:** Dependency injection [6] is a design pattern used in OOP to resolve dependencies flexibly (and to improve testing). Here a client or user of a service delegates the responsibility of providing its dependency to external code that initializes the client. This simplify the client design as it does not have to know how to construct the services. This also enhances the dynamism of the code, as the dependency might be injected anytime. In Tuscarora, many of the interfaces use dependency injection. In many cases the delegate to call when a particular event occurs is injected to the module.
5. **Functional Syntax and Messaging Semantics:** Key interfaces in Tuscarora has been designed with Message Passing Semantics. This allows the interfaces to be implemented as separate processes which allows for greater flexibility of building and deploying the framework. However, message passing semantics are somewhat counter intuitive for Interface definition and would lead to define and standardization of the messages. To avoid this and to keep development using the interfaces intuitive the function style of retained. A shim layer would be used to convert these function calls to message (and back into function calls at the other end) if messaging is needed. Else if direct binding is possible the shims would make a direct function call.
6. **Deployment/Binding Agnostic:** Tuscarora aims at a design where one module (say an ASNP or app) is agnostic to how other modules are built and deployed, and achieves binding between various modules without changing core module/class code. This principle has implications across many features such as threading, parallelism, build, deployment and communication shims. In general, multi-threaded implementations are discouraged, while multi-process implementations are supported seamlessly. Any module or interface can be implemented as a separate process and can be bound to other modules using a shim layer. On multi-core systems where many cores are available for use of Tuscarora modules, a multi-process deployment is encouraged. Although, nothing in the design prevents a multi-threaded execution of a particular module, the classic atomicity and deadlock issues that arise in multi-threaded applications needs to be handled here as well. The advice to the

developers is to avoid threaded execution unless it is critical for a particular use case. In POSIX /Linux based systems many of the Platform Abstraction Layer module events are implemented using Kernel Signaling. Handling of Kernel Signaling in multi-threaded environments is generally complex and is another consideration for using multi-threaded designs.

7. **Code Lifecycle Support and End-to-End testing:** Reliability of computer and networked systems has been falling for the last couple of decades from the standards set by the early telecommunications systems such as the telephone network. This is due to the dominance of software in communication systems, more importantly the fast evolution of software systems and the resultant complexity in testing systems in scale. To improve the reliability of networked systems a new approach is needed where code evolution, regression testing and in scale testing is co-designed from the very beginning. Tuscarora takes a lifecycle approach that maintains tight coupling between ASNP and application design, implementation, validation, and in-field operation and maintenance. This achieved by a design that keep the core module code clean and separate from the test code and running the same code in various testing regimes, such as inside a network simulator, distributed testbed or deployment. Tuscarora comes pre-integrated with the open-source ns3 network simulator.

8. **Templates and avoiding "runtime" effects:** A design where code reusability is key to minimizing bugs in implementation. Inheritance is a feature of OOP that lets developers reuse existing classes and modules. However, using simple inheritance to achieve reuse results in run time dispatch of methods which slows down execution and might result in run time errors. Templates allows a developer to achieve code reuse without 'dynamic dispatch'. Templates allow more compile time or static type checking and binding, which improves system reliability. Templates also allow another degree of flexibility where types can be specified as parameters and allows the framework to be customized to a given platform without actually rewriting any of the code. Tuscarora used templates heavily for both code reuse and portability purposes. Avoid

9. **Minimal External Dependency:** Tuscarora is a clean-slate design, not just in an architectural perspective but also from software engineering perspective. It does not require anything other than standard C++11 compiler and libraries on all supported platforms. Tuscarora explicitly avoids using any external libraries in its core modules in order to avoid dependencies for porting to new platforms. Tuscarora comes with a reasonable amount of library code for things such as Data Structures, Platform Abstraction modules, such as Timers, Schedulers, Random number generators, Serialization/Deserialization modules, communication shims such as sockets servers and clients. Additionally, it also defines interfaces for external services that might be used by Patterns and Apps that might be provided by a platform.

3.2 Core Framework Services

The framework APIs establishes a standard layer through which services are provided to the Patterns. The framework provides the following services to the Patterns, either by itself or by leveraging other existing platform services.

1. **Network Discovery:** Discovers nodes in the neighborhood, including their waveform local address, framework address and other schedule information needed for synchronous link estimation. Further, the patterns that have requested notification will be notified using callbacks, when new neighbors are available or when existing neighbors are lost.

2. **Links Estimation:** Estimates the quality of the link between the node and its existing neighbors using synchronous (with knowledge of time and schedules) link estimation techniques. Further, if a ASNP

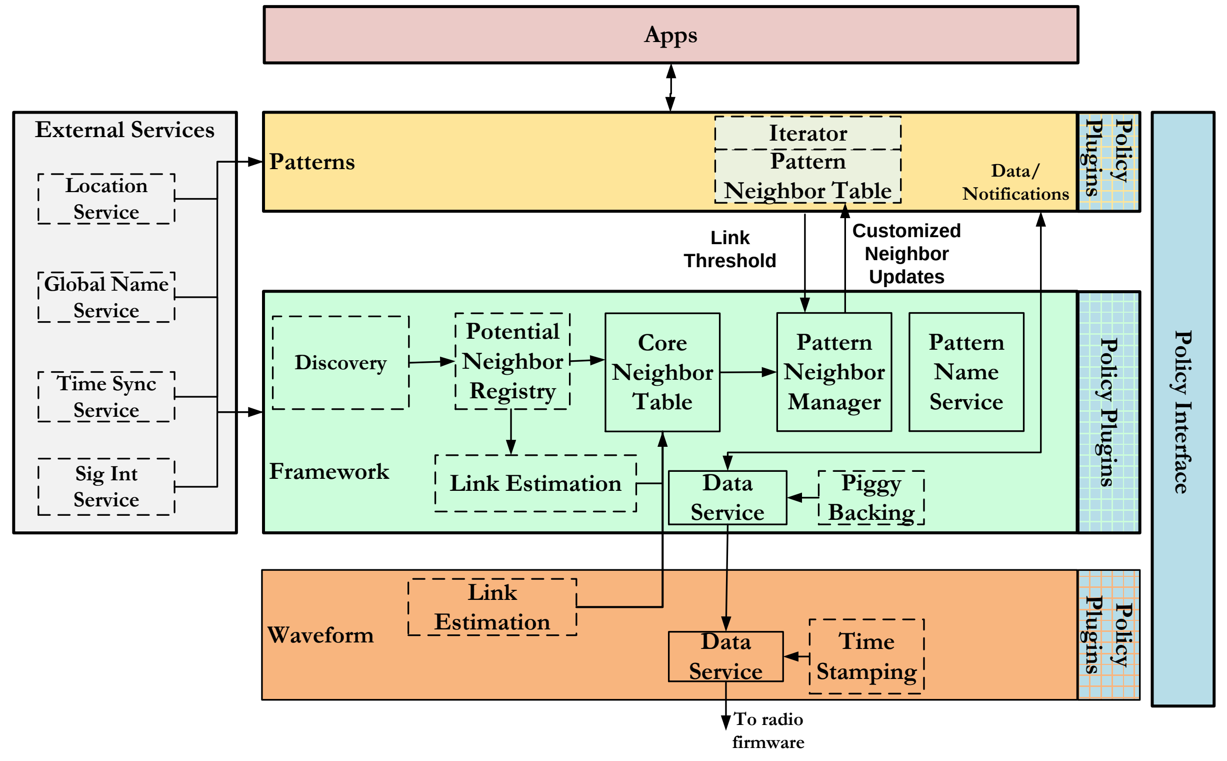


Figure 9. Deep dive of framework modules

has request support for custom link estimation, the framework will provide support to piggyback any additional information that the ASNP needs for its estimators on top of the basic estimation.

3. **Data Service:** The framework provides a data service to the ASNPs for each of the supported waveforms. The ASNPs hand over a single packet to the framework to be sent using a particular waveform to a particular neighbor (or to be broadcasted in the local neighborhood if the waveform supports broadcasting). The framework will generate an acknowledgment for each packet sent out and will notify the pattern (if requested) about the result of each packet. The framework might use waveform adapters to break up the packets or reassemble to implement the data service. A packet that has been handed over the framework, but for which an acknowledgement has not been received yet might be canceled by the Pattern. All other packets are queued up inside the ASNPs and can be manipulated freely by the patterns.

4. **Pattern Naming:** The framework provides name services which provides each instance of a ASNP running on a node with a unique ID. Every packet that is sent by a ASNP should include its instance ID, which is used to send messages to the correct target ASNP. An optional Pattern Lookup service, which provides patterns with the information of the instances and types of ASNPs that are running on each of its neighbors is also specified using Service Programming Interface (SPI) is also specified in the framework definition. However, its implementation (as the name suggest) is optional on a particular platform.

5. **Policy Management:** This service allows a network administrator to specify a set of flexible policies to manage the entire network. This service provides no direct APIs to a pattern. However, it can be thought of as providing APIs to a special management pattern that can modify and add new policies

across the network. This service is envisioned to be extensible and flexible and is separate document will be created to document its design

3.3 Modular and Configurable

Modularity has been given high priority in the design. The key challenge in the design is to provide the framework services uniformly across all supported operating platforms, irrespective of the individual capabilities of the platform. At the same time the framework would strive to make maximum use of the capabilities that a platform offers and the services provided by it. Thus a design has been evolved where clean interfaces have been defined which can be provided and instantiated either inside or outside of the framework. This also means providing a flexible way to configure modules on a given platform.

Figure 9 shows the deep dive of all the modules of the framework. Each of these modules have well defined interface which could have multiple implementations. The modules defined under the following namespaces:

1. **Application Writers Interface:** Groups classes, variables and structs that are useful for Application Developer
2. **Pattern Writers Interface:** Groups classes, variables and structs that are useful for Pattern Developer
3. **Waveform Writers Interface:** Groups classes, variables and structs that are useful for Waveform Developer
4. **Framework:** Groups classes, variables and structs that form the core framework
5. **External Services:** Groups classes, variables and structs that are not used by the framework, but which might be available on a given platform and can be used by Patterns and Applications. Defined for the sake of standardization of services and portability.

3.4 Dynamic Plug & Play of Patterns and Waveforms

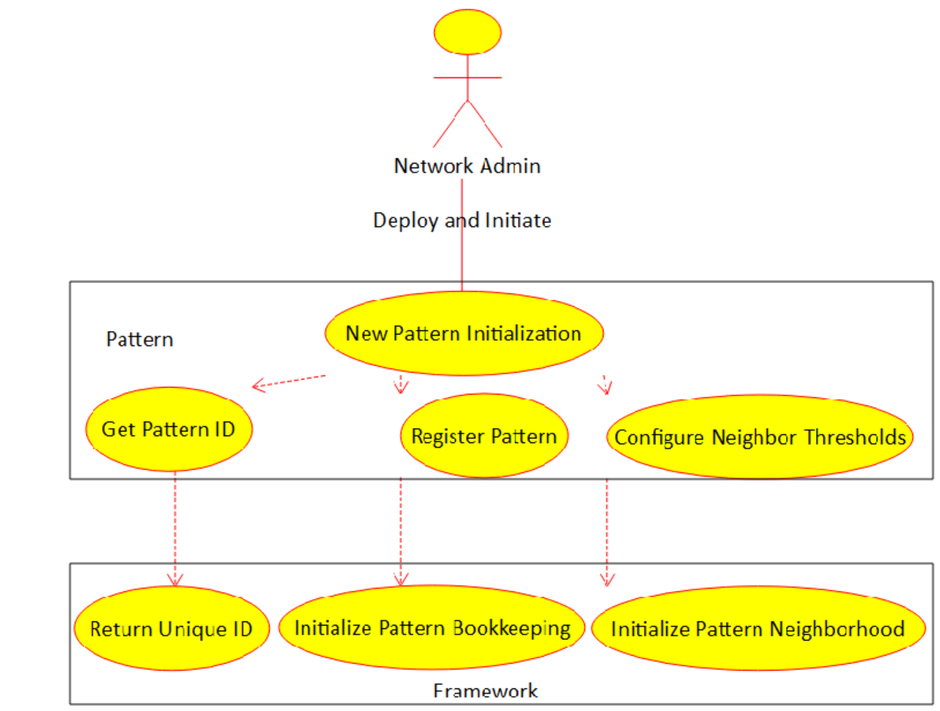

Network Admin
Deploy and Initiate
Pattern
New Pattern Initialization
Get Pattern ID
Register Pattern
Configure Neighbor Thresholds
Return Unique ID
Initialize Pattern Bookkeeping
Initialize Pattern Neighborhood
Framework

Figure 10. Use case diagram for Pattern Instantiation

Framework supports dynamic instantiation of Patterns and Waveforms. That is, the framework can be booted up without any Patterns or Waveforms, and then 1,2 or any number of Waveforms and Patterns can be booted up separately and they can register with the framework. While, a waveform can "connect" at any time with the framework, it still has to be one of the waveforms, that the "framework" is aware of. This dynamic plug and play is more of an operational feature and does not means "any arbitrary radio" can be plugged into the framework at any time, without a-priori knowledge about the waveform features. However, we do expect more dynamism on the Pattern side, since a Pattern is "more of a client" to the framework any Pattern developed on top of the Framework APIs can be dynamically instantiated at runtime, as long as the Policy Manager/Network Administrator is aware of the Pattern.

Figure 10 show the (UML) use case diagram shows the instantiation of new pattern. The use case diagram for instantiating a waveform is similar.

3.5 Platform Independence

Being independent of the deployment platform is one of the key goals of Tuscarora.

This is achieved using several design principles as explain in **Section 3.1** in design principles 6-9.

One of the ways to achieve platform independence is to abstract out all changes that could affect ones' software. This is usually done using what is known as a Platform Abstraction Layer (PAL). A PAL defines all modules/APIs that are required by the software from the under laying hardware/OS and leaves the implementation of these modules to the specific platform. All modules that use only these PAL APIs (and each other) are thus isolated from the PAL and requires no changes.

A user porting the framework to a new platform will need to provide the PAL modules on that platform, thus porting the framework in essence is just porting the PAL.

3.6 Platform Abstraction Layer Design

The design of a PAL needs to be considered carefully.

Two designs were considered, one based on Interfaces and another based on what is known as a "Pointer to an Implementation" (or PIMPL in short) based Invalid source specified.. The primary requirement of a PAL class is that it allows a single definition and multiple implementations. This can be achieved using Interfaces (a pure abstract class that cannot be instantiated) in OOP languages using inheritance and polymorphism, by defining an interface for each PAL module and having multiple implementations for each platform.

However, this has a drawback of not being able to instantiate the Interface and therefore on each platform the user will need to instantiate a different implementation which does not achieve portability. An improvement here would be to #ifdef the code heavily for each platform. This would still not achieve desired result, since each time one ports to a new platform new #ifdefs need to be added to code.

PIMPL provides an alternate solution to this. In PIMPL the abstraction is just a regular class (not pure-abstract as in an interface) and therefore can be instantiated. It just hides the implementation details from the abstraction class and lets us have multiple implementations (or cpp files) in say different platform depended directories. PIMPL does have some drawbacks such as another function redirection in many cases and non-availability of private members in derived classes, but these are reasonable tradeoffs to achieve platform independence. At this point a user might be wondering "why do we even need the PIMPL design pattern", and "why is just a plain class definition not sufficient?" The reason is that c++ does not support "partial definitions of classes" and once a class is defined for one platform it can't be modified for another.

3.7 Deployment Agnostic

Tuscarora is targeted for 3 key deployment architectures which should cover over 90% of computing devices:

1. Big Metal: Multi-core/Multi-SoC Server type systems: Software and radios from many vendors are running possibly as different processors, may be even on dedicated cores.

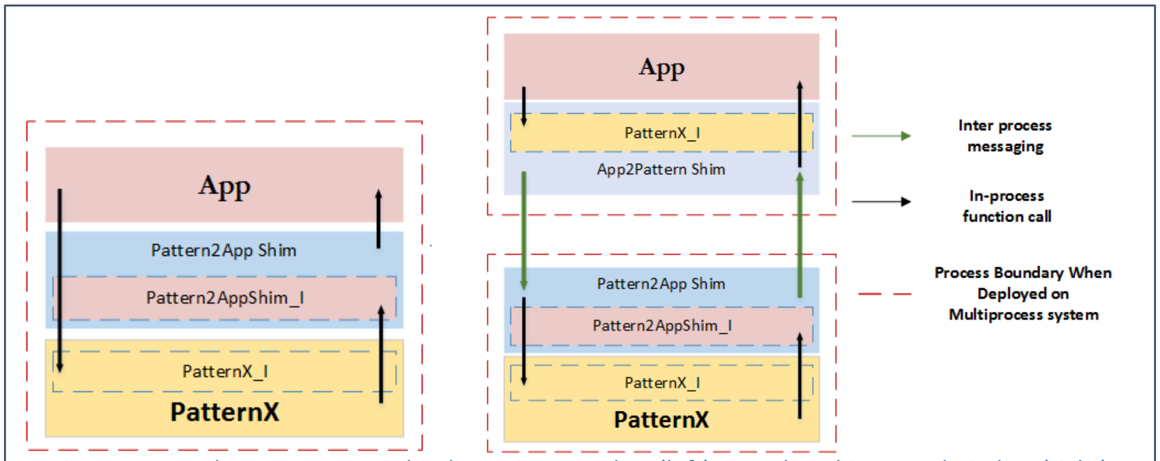


Figure 11. Deployment agnostic binding: Direct Binding (left) vs Socket Shim Based Binding (right)

2. PC/Smartphone class: Built as a single executable
3. Bare-metal: No separate underlying OS, no file system. Built as a static library to be linked with rest of the software to be run in single address space.

Tuscarora supports deployment agnostic binding for its key modules. That is, a Framework, Waveform, Pattern and App can either be built and deployed as a single process on a single node or as many processes running even on a network and can work seamlessly. This provides great flexibility in deploying Tuscarora on legacy systems and on big metal systems.

Two key design features enable this:

1. Messaging Semantics and Functional Syntax explained in **Section 3.1** .
2. remote binding using communication shims.

3.8 Communication Shims

Figure 11 shows how this would work in a single process binding or a multi-process binding.

A communication shim is layer that exposes a remote interface to a local module as though it is on the same process and transparently takes care converting the local function calls to messages to remote process and vice versa. The communication shim itself can be achieved in several different ways such socket based, pipe-based, file IO, RPC, SOAP, etc. The appropriate shim can be chosen based on target platform(s). Tuscarora comes with a general purpose socket based shim to support deployment on legacy systems.

3.9 Minimizing message copying between modules

The core of any networking framework is the passing of messages structured as headers and payloads among various modules and layers, finally sent out on the radio (or wired) channel to be received on one or more neighbors.

Networking frameworks usually end up making a lot of copies of the same message as a message passes through the network stack layers and into the Kernel and out through the radio. This decreases the performance of the system. Tuscarora is designed to minimize copying of messages as long as they are passed inside of the same process.

This is done by two features:

1. separating out the payload of a packet from the its header (headers are much smaller than payloads and hence is ok to sometimes copy them),
2. , passing messages between layers by references and not as value type (which forces copying). The pass by reference however complicates the memory ownership model and the assigning of responsibility of destroying the message. Tuscarora currently uses an “implicit transfer of ownership” model across the layers and receiving layer can then make decisions about the message. However, we are working on adding a more flexible and explicit ownership transfer model.

3.10 Time Synchronization/ Time Stamping of Packets in Waveform using Piggybacking

Synchronization of time across the network is problem that is common in many of the wireless networks that are resource and/or energy constrained. In general, synchronous communication can be much more efficient both in terms of channel utilization and energy usage, however this requires that the two nodes that are to communicate (the whole network in general) have a common notion of time. This can be achieved by having reliable access to some Global clock such as in GPS. But in the absence of such a clock usually nodes are synchronized using what is known as a Time Synchronization (TimeSync) Protocol. Such protocols can either be part of radio/MAC/Waveform layer or outside of it.

A common feature of these protocols is that they use the message itself as a synchronizing event between the nodes. There are two types of such protocols;

1. Implicit synchronization: Once a node has a rough estimate of its neighbor's time, it can use the message receive times from such neighbors to further fine tune the sync. Achieving rough estimate to begin with might need a different protocol and explicit messaging and is outside the scope of this document. One method though is explicit synchronization discussed next.
2. Explicit synchronization: A node sends its time corresponding to a message, timestamped just before the message was sent over air. And the receiver timestamps immediately after receiving it and therefore achieving synchronization except for over the air delay.

Both 1&2 requires timestamping on receiving side and dispatching these timestamps to a TimeSync protocol module. In addition, explicit sync also needs timestamping on the sending side. Both sending and receiving timestamps are best implemented in the waveform layer.

Tuscarora supports timestamping at waveform layer for both sending and receiving. Additionally, the sending side timestamping can be done piggybacked on packets that are already scheduled to be sent and hence no additional overhead will be incurred for this feature. This feature can be configured while building the framework and waveform.

3.11 Instantiating Link Estimation

In a sense the core service of the framework is the Neighborhood service which detects neighbors, estimates their quality and notifies the Patterns. This provides a standard interface for implementing the patterns. Tuscarora will provide estimates for neighbors if the waveform does not provide them.

Tuscarora supports three types of link estimation protocols, one of which can be chosen at run time:

1. Periodic: A basic protocol that beacons periodically using a pseudo-random schedule at a given rate and has no knowledge of its neighbors' beaconing schedules. Links are removed after a period of time (called the Inactivity Period) goes by without receiving a beacon. This parameter is specified by a positive integer > 0, which is interpreted as a multiple of the beaconing period; the default value is 3.

2. Schedule-Aware: This protocol beacons periodically and maintains a record of neighbor's beaconing schedules. Links are removed when a beacon should have been received, but has not been.

3. Conflict-Aware: This protocol also beacons periodically and maintains a record of neighbor's beaconing schedules. In addition, it considers potential conflicts in the beaconing schedules of its neighbors. Links are removed when a) an expected beacon is not received, and b) there is no conflict with

the schedules for the other nodes in the Neighbor Table. If there is a potential conflict, the link is not removed, but link quality does go down.

There are five link metrics supported for each link:

1. **Quality:** Abstract quality metric expressed as a real number between [0,1]
2. **Data rate:** Link data rate expressed as log2 (bps)
3. **Average Latency:** Average latency in sending a packet to the destination, expressed as log2 (seconds)
4. **Energy:** Average energy to transmit a packet expressed as log2 (picojoules)
5. **Cost:** Abstract average cost of sending a byte

The estimation protocols together with the framework will measure these metrics and will publish them to the patterns. The critical metric is the *Quality* metric that assigns a numeric value between 0 to 1 to each of the neighbor's indicative of its stability and channel quality. **Figure 12** shows the link estimation flow from the waveform into the framework and to the patterns.

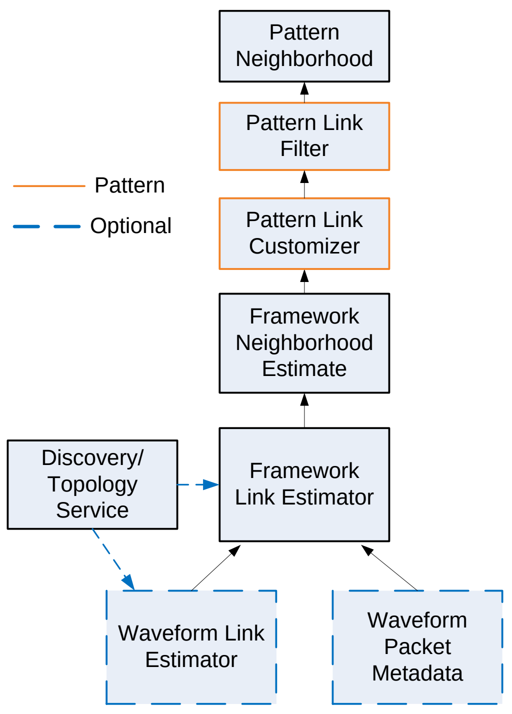


Figure 12. Link estimation work flow from waveform to patterns

The three link estimation protocols differ mainly in how they interpret whether or not a particular estimation message is received correctly. They all use the same 'beaconing' mechanism. (Beaconing is a term usually used to indicate a periodic local broadcast message.)

The modular design of the framework (and the waveform) lets a user to configure and instantiate, any of the above there link estimation protocols as part of the waveform layer. If these are not instantiated at waveform and if the waveform does not have its own estimation, then the framework detects this and will instantiate a default estimator (set through the policy manager) for that waveform. Please refer to **Figure 9** for system module details.

3.12 Policy and Administration Manager Interface: Interface for configuration and run time administration

Tuscarora currently supports a simple, but functional policy interface. The policies will need to be specified when the framework boots up. Dynamic run-time policy specification is under development and will be supported in future. Currently the framework supports four groups of policies, namely:

- **Pattern Policies:** Policies about a Patterns priority and resource usage.
- **Link and Packet policies:** Policies about link selection (if multiple links to a node are available) and the frameworks packet buffering and scheduling.
- **Network topology policies:** Policies that govern network topology, like adding radio silent nodes or black listed nodes.
- **Framework features:** Policies about enabling or disable certain framework features.

A list of currently available Policy and Administration APIs is listed in the table below.

Method	**Function**
bool SetPacketPolicy (PacketSchedulingPolicyE _policy)	Specifies the policy for outgoing packets. The default policy is pattern priority based. First Come First Served (FCFS) is another supported policy.
uint16_t GetAsnpPriority (uint16_t asnpInstanceId);	Get the priority of a pattern instance.
bool SetAsnpPriority (uint16_t patternInstanceId, uint8_t priority);	Set individual pattern priorities, if the policy is PriorityFCFS. Has no effect if PacketPolicy is FCFS.
bool AddNode (NodeId_t id, NodeInfo info);	Add a node not discovered by framework, like special nodes.
bool AddRadioSilenceNode (NodeId_t id, NodeInfo info, double lat, double long, uint64_t radius)	Node should be added to the network. Any node which is inside this circle will see this radio silence node in its neighbor table
bool RemoveNode (NodeId_t id)	Remove a special node from network.
bool SetBlackList(NodeId_t *nodeArray, uint16_t noOfNodes);	Blacks lists the nodes passed as parameters. The black listed nodes will be removed from neighborhood even if they are detected.
bool ClearBlackList();	Clears the current black-listed nodes
bool IsPiggyBackingEnabled()	Returns the status of piggy backing in the framework
bool IsTimeStampingEnabled()	Returns the true if TimeStamping is enabled at waveform layer.

3.12.1 Envisioned Future Design

The previously described existing policy manager is functional and suitable for most initial deployments.

However, to accelerate future development, and to enable users to make their own adaptions, Samraksh envisions a policy manager for Tuscarora that is extensible and flexible. While the primary user of the policy interface currently is a system integrator, an equal role is being envisioned for a mission planner or network administrator who would like to query the system modules for its supported policy interfaces rather specify them at run time.

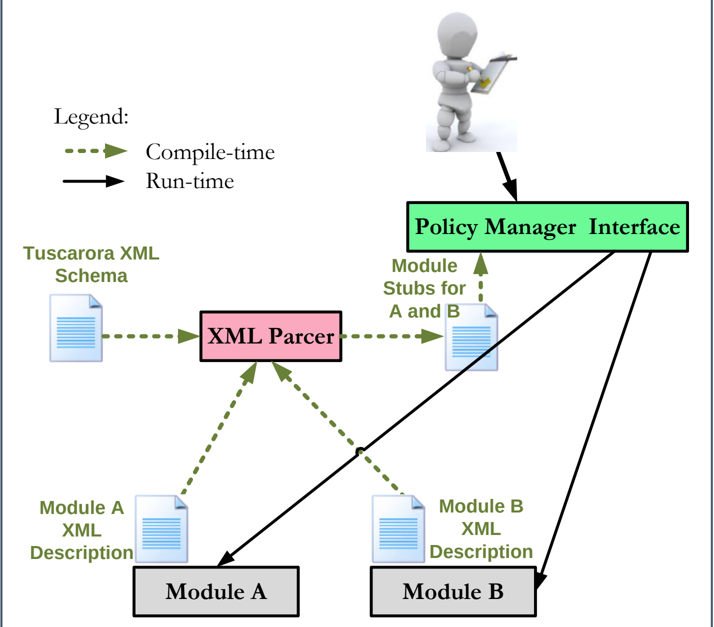


Figure 13. Envisioned extensible policy manager design

Consequently, an architecture is envisioned where each module developer provides a management interface for their module along with the primary interface. And the management interface can evolve much faster (and even between missions/operations) than the main interface. Such a design is shown in Figure 13. The current snapshot of the management interface would get exported in some machine readable format (such as XML). The policy manager would then query the modules of interest and would aggregate the policies and expose them to a Network Manager—possibly a human—to set the policies. While such a design is envisioned, it is not currently under active development. Users interested in such a manager can contact Samraksh developers to collaborate on its implementation.

4 Case Study: Futuristic Forward Operating Base

4.1 A Prototypical Mission

In this hypothetical near-future example, a Marine Special Operations Command (MARSOC) platoon is deployed to a small remote Forward Operating Base in a middle-eastern country, and is tasked with securing the FOB, performing Theater Security Cooperation (TSC) with a detachment of host nation troops, and gathering Intelligence, Surveillance and Reconnaissance (ISR). They will be conducted both mounted and unmounted Distributed Operations from this FOB.

4.2 Assumed Technologies

In this futuristic use case we assume a number of technologies that we believe are already available or might be available in near future.

1. **Footfall Sensor:** This is a seismic sensor. Their power usage is low, but the sensor contains a comparatively heavy mass, which makes them a relatively heavy compared to other sensors. The range varies from about 15 m to 60 m depending on the soil type. They only detect humans walking normally, and have some trouble suppressing detections from machinery in industrial areas.
2. **Acoustic Sensor:** Acoustic sensors are low power, but require more computing power than other sensors. This makes medium power sensors; however, because the packaging is so simple they are the lightest sensor. They can detect quiet passenger cars at 40 m and military trucks at longer range. These sensors can also detect small arms fire. A small cluster of them can recognize that a sniper (or other combatant) took a shot in their region, and when the shooter is inside the field pinpoint the shooters location.
3. **Anti-SOF Radar:** The radar is a relatively high power sensor. It able to detect slow moving targets crawling on the ground. Its range is only about 15 m. It has the potential to evolve into the sensor with the lowest false alarm rate.
4. **Mine Alternative Device:** These nodes contain an explosive, a footfall sensor, an acoustic sensor, and anti-SOF radar, and a short range camera. They detect intruders, capture a picture, and send it to an operator, who can decide to detonate a portion of the field.
5. **Short Range Cameras:** These are similar to the night vision cameras that can be bought online, except that they are attached to the network so that a few images or a short video clip can be sent to the operator. They can capture operationally useful images up to about 50 m. (This is a short range compared to current military cameras, but these are smaller and much lower power). The consumer grade devices use Pyro-Electric IR (PIR) motion detectors to turn the camera on only a small portion of the time. In this near future scenario all the other sensors will activate the camera. The battery specified in Table 1 assumes a 7.5% duty cycle.
6. **Micro-UAVs:** These devices can be directed to watch a specific spot on the map. They will then fly random patterns around that spot taking video of that spot and transmitting a sequence of individual frames from the video back to the operator's tablet. The operator can scroll back and forth though stills shots and ask for full speed video of specific events or times. When the UAV returns to be recharged the operator can plug it into the tablet and quickly save all of the video. The UVA will

automatically transmit additional frames if the image changes more than is typical. The operator is not expected to directly fly the micro-UAVs, which would be incompatible with fighting.

7. **SIGINT Sensor:** This sensor uses a Linux class embedded computer, a Digital Signal Processor (DSP) board, and a wide band front end. As a result, it is expected to consume approximately 250 W. In this scenario a Li Ion battery is proposed because they are lighter than most alternatives. Assuming a specific energy of 160 Wh/kg, 20 kg battery would provide 12.8 hr of contentious operation. Solar cells are required to provide operating power and to recharge the battery during daylight

4.3 Use Cases

The following use cases are highlighted within this scenario -

4.3.1 Perimeter Defense

The platoon distributes rings of remote sensors around the perimeter of the camp, projecting their network far out into remote "over the horizon" locations throughout their assigned Area of Operations (AO), including on surrounding high points from which an attack might be made on the FOB. The distribution of the varying kinds of devices is done very simple, with light camouflaging, and the different devices automatically configure into an infrastructure-less mesh network, with multiple paths back to the base camp.

Back at the FOB, sentries are able to monitor the sensors. SOF radar sensors that report potential human presence can automatically trigger the launch of micro-UAVs, slew advanced cameras to focus on the area, and alert human observers to these events and data. The human attended event can now also use the meshed network connectivity to decide whether or not to arm land-mine-alternative devices, based on positive identification of hostile threat. This prevents land-mine detonations to innocent travelers, or unintended destruction of landmine assets from wildlife.

As a Force Protection Kit, the solution does not require significant dedicated networks, heavy expensive dedicated proprietary hardware. It allows a limited force of fallible humans to significantly and effectively expand their protected perimeter, while balancing the abilities of autonomous action with human oversight, all while reducing the stress and strain of constant surveillance on the troops. This technology thus becomes a force multiplier.

4.3.2 Direct Action

Squads from the platoon can be assigned to "observe and monitor" direct actions from host troops onto confirmed hostile sites. Traveling in a combination of military and civilian vehicles, they can maintain constant communications with a mesh of UAVs and marines on foot, across rugged terrain. Establishing multiple "over-watch locations", they can use their Nett Warrior devices, Headsup display, and rifle scopes mounted to Intelligent Rail systems to share data, as they watch the attack by host nation troops on enemy positions. That includes:

- All intelligence is networked into a Common Operating Picture (COP), available to all human or machine assets in the action. This Tactical Internet can be fed directly into the Global Information Grid (GIG). This allows tactical flexibility for course of action best suited for an evolving conflict, within a comprehensive Distributed Operations integrated communications and COP. COC and

Indirect fire agencies are automatically updated, and not subject to verbal updates by COs via their VHF or other voice communications platforms.

- Communications parsimony. Rather than having all nodes flooded all of the time with RF set to maximum capacity, the RF links are limited in power to just the amount needed to reach the appropriate nodes in the network for the application being communicated (thus lowering RF detectability, improving battery life), and traffic only reaches the nodes that it is needed on. Interoperability of multiple radios minimizes the complexity of the growing number radios that each team needs to carry.
- Automatic "blue force tracking" that paints positive clear images on all rifle sights of friendly forces, include the Multi-Purpose Canine (MPC).
- POIs or digital "Chem-markers" to be designated. A person or asset tagged as high value by one marine would then show up clearly on the displays of the other marines.
- Autonomous action by the UAVs could be directed in part by the location of blue forces, designated POIs, as well as by the current focus of marines. This focus would include the real-time locations of the focus area of each rifle scope in the squad at the time, allowing for high quality artificial intelligence allocation of surveillance such as the UAVs
- Realtime status of the individual marines can be communicated to other vested stakeholders. This includes
    - vitals being transmitted to medics and nearby marines and on the ground commanders, ammunition count
    - ammunition count (both in rifle and in direct proximity to each marine) could be communicated not only to other troops, but to autonomous unmanned ground vehicles that can prioritize and resupply the troops in the field, automatically
    - Marine readiness (hydration, stress level, awake/sleep) could be monitored by local platoon sergeants and Cos, for better real-time readiness and situational awareness

4.3.3 Logistics Resupply

Logistics inventory on hand would be automatically available and updated in realtime.

Squads low on ammunition, for example, due to an ongoing firefight, would trigger not only local resupply by UGV, but meshing back to command vehicles, FOB and company staff at HQ, logistics resupply for the FOB could be triggered.

Crates being delivered by helicopters to the FOB would mesh out and provide real-time updates as to the inventory they contain. This could be done via active ultra-low-power embedded RF mesh application updated for the ASNP architecture, integrated into an enclosed military UNISEG/Pelican type ruggedized pallet to provide true assured end to end enhanced in transit visibility (EITV) access and transparent identification, authentication, and content monitoring across the network as it moves end-to-end through shipping supply chains. This allows more rapid and efficient resupply to those that need it, with full oversight and manageability by those tasked with managing the resources.

4.3.4 MEDCAP

Medical outreach to indigenous residents of nearby villages, could mesh together and network medical equipment for better diagnosis and care of villagers, while simultaneously relaying any Intelligence gained in the village back out to those assets that need that information in realtime.

Vehicles delivering medical supplies would provide real-time inventories to medics who might need to know if a particularly medicine was on hand.

A patrolling squad could quickly deploy a temporary sensor mesh networks around the perimeter of the village during the MEDCAP mission, to help secure the troops and villagers from ambush. At the end of the mission, the sensors are picked up (they provide constant GPS independent Location Based Information to the platoon, so they are easily located)

4.4 Advantages of using ASNPs/Tuscarora

Tuscarora-based networking allows well over 1,000 nodes of diverse hardware type and radio waveforms to network together in ad hoc fashion, using routing that not only effectively scales to meet this large, diverse and remote cloud of devices, but also routes traffic optimally across the network based on the needs of many different applications. This creates a range of benefits to multiple stakeholders:

- To the Marine Corp, (the organization procuring and deploying the technology) this enables a broad range of use cases not possible today, creating emergent properties that create compelling strategic advantages, including Manned & Unmanned Teaming (MUMT), and Third Offset strategies. E.g:
  - Enhanced real-time connectivity creates the opportunity for “real-time, big picture, situationally hyper aware Artificial intelligence” for remote stakeholders
  - Widely connected computing assets creates a framework against which distributed edge computing can allow local computing assets to be coordinated into collaborative computing processes (with lower latency, better energy efficiency, and less RF of more wide-area-distributed computing architectures)
  - Steady reliable communications that empowers the Network-Centric Warfare (NCW) doctrine. Functional in crowded bandwidth situations. More robust communications than previously available over longer distances then previously available with portable radios that last longer than previously available (for years) on internal batteries
  - Ad hoc networks allow “over the horizon” force projection, protection, and intelligence gathering
- To the Marines who are the end-users – they are safer, more effective, (through enhanced connectivity) and have less of a logistics burden to manage, carry and services, since the open-source ASNP effectively displaces heavy, expensive proprietary radios.
- To the hardware vendors enabling these scenarios – they now have a solution to their customer’s needs, without having to invest in proprietary technologies – a process which is slow and capital intensive.
- Additionally, there are a number of discrete specific advantages which accrue to stakeholders whose roles in the mission are less obvious, and easy to over-look, but still mission critical. E.g:
  - The use of ad hoc networking can be tailored to minimize radio footprints, (creating better radio discipline), by daisy-chaining less powerful signals between nodes on the network to make long distance communications possible, minimizing the RF signature of the troop.
  - Logistics and resupply stakeholders – are kept continually informed of the real-time status of inventories and needs of critical supplies, facilitating efficient and safe distribution of logistics in ways not possible today.
  - Hyper efficient use of communications stands to have a profound positive effect on battery life, reducing costs, supply chain complexity, and field issues

- Redundancy of communications pathways between assets creates a very strong resiliency within the communications. E.G. avoids critical failures in the case of bullet through a rifleman’s radio (because now is rifle scope, physiological state and other meta-data is still being transmitted via other nodes

4.5 Use Case Chart

The following chart summarizes the use cases possible under this mission what the advantage of using the ASNP architecture over the traditional approach

Use case	**ASNP**	
	Description	**Advantage of ASNP**
Common Operating Picture	Data (e.g. intentional communications) and meta data (e.g. telemetry) from the unit’s assets available to commanders, others, and computing assets/vehicles for unified situational awareness	Variety of kinds of devices, using a variety of radios, can be meshed together ad hoc, in very large numbers. RF signatures are reduced, & battery use lengthened.
Force Protection	Lots of cheap, small assets provide 24/7 reliable force protection	Mesh allows low RF, low battery, high resilience in uncertain terrain
Swarming UAVs	UAVs communicating with each other (P2P) UAVs sensing meta-data from ground (POIs as determined by leaders, POIs as inferred from weapon orientations) UAV real time status (e.g. fuel status, telemetry, data) surfaced to appropriate human assets	Direct Peer to peer connectivity with low latency allows AI driven “swarming” behaviors that optimize autonomous use of multiple assets. Meshing with human and other assets allows better target prioritization
Smart Guns	Weapons platforms wirelessly connected to rest of individual warriors’ kit, AND to directly to other devices in squad/unit	Allows dynamic on-scope rendering of POIs. Transmits FoV to others Renders Blue Force data Allows meta data (ammo count) to get to right assets
Intra-unit communications	Intentional and telemetric data transmitted within the unit	Allows minimal RF to be used to transmit data. No line of site, minimal detection, more resilient to contested situations
UGVs	Communication (directed and autonomous) between UGVs to human and other machine elements of the squad	
Soldier metadata	Data not typically communicated today: ammo count, HR, stress level, hydration, awake/asleep	Made available into local and remote assets: command staff, other human unit members, machine assets running autonomously

5 Bibliography

[1] "Tuscarora: A framework for developing distributed applications for hetergenous networks," The Samraksh Company, [Online]. Available: https://Samraksh.com/index.php/products/tuscarora.

[2] "Tuscarora Source, Tools and Releases," [Online]. Available: https://github.com/Samraksh/Tuscarora.

[3] D. L. Alderson and J. C. Doyle, "Contrasting views of complexity and their implications for network-centric infrastructures," *Systems, Man, and Cybernetics Society,* vol. 40, no. 4, pp. 839 - 852, July 2010.

[4] "CMake: An open source, cross-platform build system," [Online]. Available: https://cmake.org/.

[5] "Direct Code Execution," [Online]. Available: https://www.nsnam.org/overview/projects/direct-code-execution/.

[6] "Dependency Injection," [Online]. Available: https://en.wikipedia.org/wiki/Dependency_injection.

Appendix A Acronyms

Acronyms	Definition
AO	Area of Operations
API	Application Programming Interface
ASNP	Application Specific Networking Pattern
ASNPA	Application Specific Networking Pattern Architecture
CO	Commanding Officer
COC	Command Operations Center
COP	Common Operating Picture
DARPA	Defense Advanced Projects Agency
EITV	Enhanced in Transit Visibility
FCFS	First Come First Served
FOB	Forward Operating Base
GIG	Global Information Grid
IP	Internet Protocol
ISR	Intelligence, Surveillance and Reconnaissance
MAC	Medium Access Control
MPC	Multi-Purpose Canine
MUMT	Manned and Unmanned Teaming
NCW	Network-Centric Warfare
NIC	Network Interface Card
OOP	Object Oriented Programming
OSI	Open Systems Interconnection
OTA	Over the Air
PAL	Platform Abstraction Layer
PIMPL	Pointer to an Implementation
PIR	Passive Infra-Red
RPC	Remote Procedure Call
SIGINT	Signal Intelligence
SOAP	Simple Object Access Protocol.
SOF	Special Operations Force
TCP	Transmission Control Protocol
TSC	Theater Security Cooperation
UAV	Unmanned Ariel Vehicle
UDP	User Datagram Protocol
UGV	Unmanned Ground Vehicle
UML	Universal Markup Language
VHF	Very High Frequency
XML	Extensible Markup Language